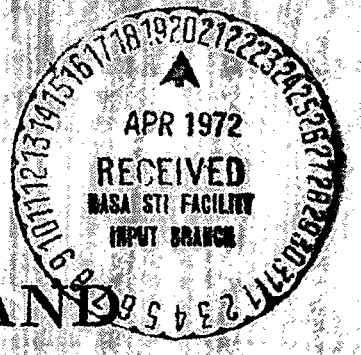
Technical Report TR-170
NGR-21-002-197

September 1971

Architectural Design and Simulation of a Virtual Memory

by

Gee-yin Kwok and Yaohan Chu



UNIVERSITY OF MARYLAND
COMPUTER SCIENCE CENTER
COLLEGE PARK, MARYLAND

Technical Report TR-170 September 1971
NGR-21-002-197

Architectural Design and Simulation of a Virtual Memory

by

Gee-yin Kwok and Yaohan Chu

This research was supported by Grant NGR-21-002-197 from the National Aeronautics and Space Administration to the Computer Science Center of the University of Maryland.

Abstract

Virtual memory is an imaginary main memory with a very large capacity which the programmer has at his disposal. It greatly contributes to the solution of the dynamic storage allocation problem, a problem of great importance in time-sharing computer system. This report presents the architectural design of a virtual memory which implements by hardware the idea of queuing and scheduling the page requests to a paging drum in such a way that the access of the paging drum is increased many times; for the design reported here, an increase of up to 16 times in page transfer rate is achievable when the virtual memory is heavily loaded. This in turn makes feasible a great increase in the system throughput. Detailed design of the virtual memory and simulation of the paging drum channel have been reported previously; therefore, only the design and simulation of the input-output control system for paging is presented in this report.

Table of Contents

Architectual Design and Simulation of a Virtual Memory

1. Introduction

Virtual memory, as is being called now, is an imaginary memory with a very large capacity which the programmer has at his disposal, even though the computer has a relatively small main memory. It is first introduced as a one-level memory on the Atlas computer by the group (11) at Manchester, England. Since then, it has been implemented by a combination of hardware and software in a number of so-called time-sharing computers such as RCA Spectra 70/46, IBM 360/67, and GE 645. Virtual memory has become one of the most important advances in the architecture of modern computer systems. It contributes greatly to solving the problem of dynamic storage allocation. Though the virtual memory is not without its problems such as fragmentation and thrashing, more knowledge and more experience have been accumulated. It enhances significantly the cost-performance ratio. It might not be imprudent to predict that most of the large-scale and medium scale computer systems in the near future will have a virtual memory, even when they are batch computer systems, because of the potential impressive improvement in cost/performance ratio.

The design in this report is based upon the virtual memory system presented by O. R. Pardo in an earlier report (6). This design makes use of the ideas reported by Weingarten (12), Denning (13,14) and Coffman (3). The central idea is to queue and schedule the page requests to a paging drum in such a way that the access of the paging drum is optimized. This idea can give a multifold increase in the page transfer rate and in turn, can significantly improve the cost-performance ratio of a computer system.

This idea, however, could not be successfully implemented by using software alone on an existing computer system. This report presents an implementation by hardware. For this particular design, an increase of up to 16 times in page transfer rate is achievable when the virtual memory is heavily loaded. More detailed information about the design and simulation is available in references (5,6).

2. Configuration of the Virtual Memory

The configuration of the virtual memory is shown in the block diagram of Fig. 1. There are six system units:

(a) central processing unit (CPU)

(b) main memory (MM)

(c) translation memory (TM)

(d) page table memory (PTM)

(e) paging drum memory (PDRUM)

(f) paging drum channel (PDC)

In this section, the translation memory is described in some detail, while the other units are treated briefly since they are described elsewhere (5,6).

2.1 Central Processing Unit

A computer system with a virtual memory allows the use of virtual address in the programs. The central processing unit fetches an instruction and translates the virtual address in the instruction into a real address by means of the translation memory. The virtual address format for the virtual memory described in this report is shown in Fig. 2. There are five fields in the format. The 1-bit D field when 0 indicates a virtual address; else, a real address. The 1-bit I field when 1 indicates indirect addressing; else, direct addressing. The 2-bit X field specifies which index register is used. When X=0, no index register is used. When X=1, 2, and 3, index registers 1, 2, and 3 are used respectively. The virtual address consists of two fields, a PAGE field and a WORD field. The 10-bit PAGE field contains the virtual page address, while the 10-bit WORD field contains the virtual word address. The

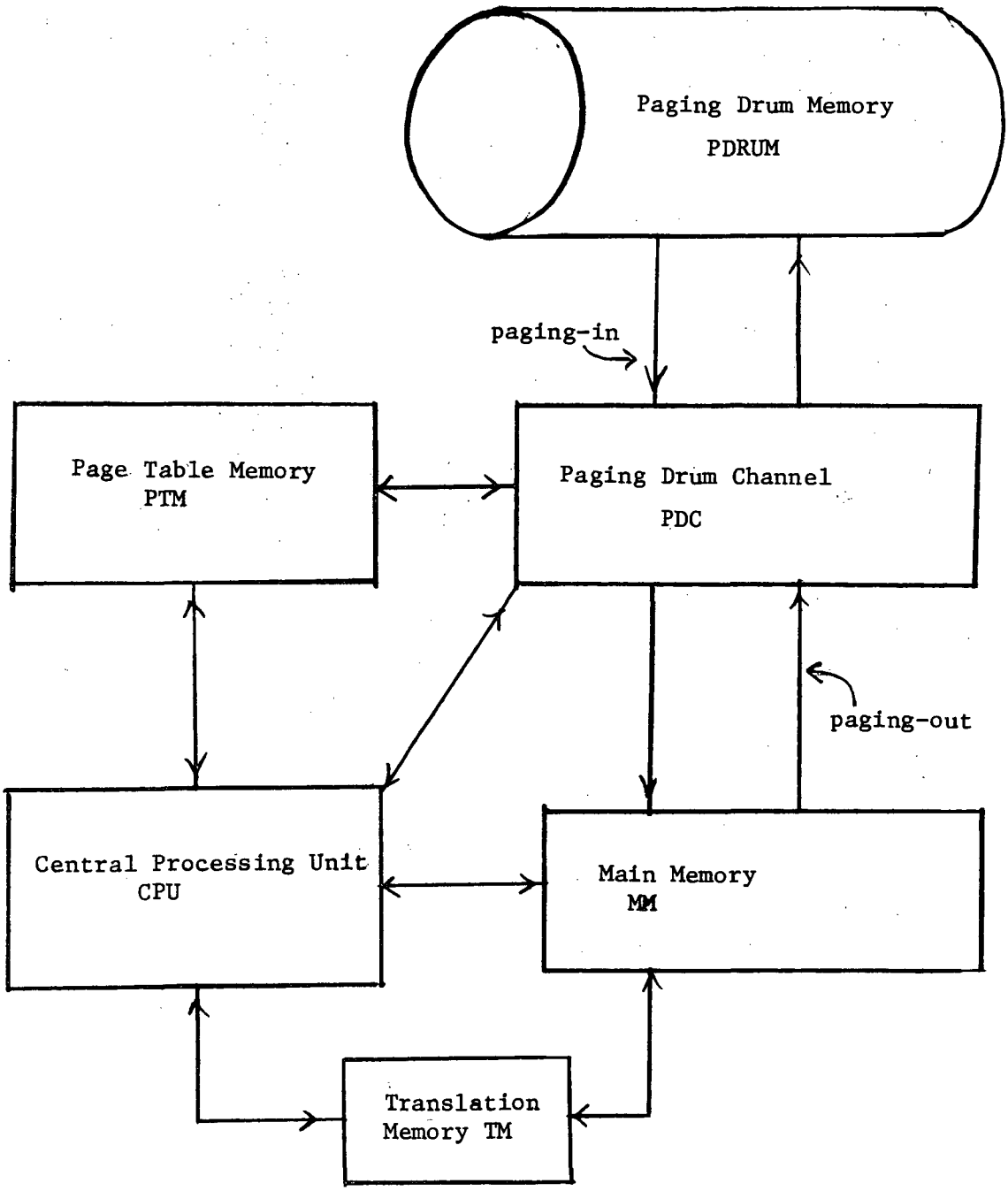
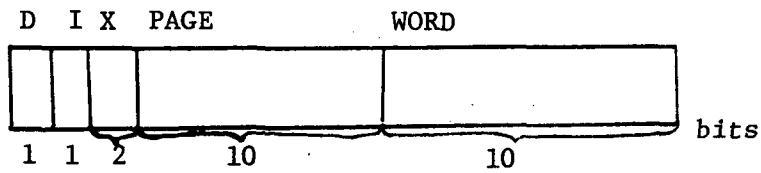


Fig. 1 Configuration of a virtual memory



D=0, virtual address; else real address

I=1, indirect addressing, else direct addressing

X=1,2, or 3 index register

PAGE virtual page address

WORD word address

Figure 2 Virtual memory address format

virtual word address is, however, identical to the real main memory word address.

In addition to instruction decoding and execution, the CPU is also capable of accepting priority interrupts.

2.2 Main memory

The main memory is assumed to have a capacity of 65,536 36-bit words. It is organized into equal size blocks of locations known as page frames. Each block of main memory locations consists of 1024 words; thus, there are 64 page frames. The main memory address is 16 bits; 6 bits specify the main memory page address and 10 bits specify the word address within the page. The characteristics of the main memory are summarized in Table 1.

2.3 Translation memory

The translation from a virtual page address into a main memory page address is achieved by the address translation table stored in the translation memory. This memory has a capacity of 1,024 16-bit words and a memory cycle time of 100 nanoseconds. The configuration of the translation memory is shown in Fig. 3. There are a 10-bit translation memory address register TADR, a 16-bit translation memory buffer register TMR, a 24-bit virtual address register VAD, and five single bit registers (TMA, RW1, RFLAG, WFLAG, and PFAULT). Register TMA is set to 1 when the translation memory is accessed. Register RW1, when 1, indicates a write operation; else, a read operation. Register RFLAG is set to 1 when there is a read error, while register WFLAG is set to 1 when there is a write error. Register PFAULT is set to 1 when there is a page fault (i.e., the CPU encounters a page not in the main memory).

The translation word format is shown in Fig. 4. There are five fields.

Table 1 Characteristics of the Main Memory and the Drum Memory

Characteristics	Main Memory	Drum Memory
memory cycle time	1 microsecond	--
memory transfer time	--	1 36-bit word per microsecond
data transfer width	36 bits or 1 word	36 bits or 1 word *
data units	(a) 36 bits per word (b) 1K** words	(a) 36 bits per word (b) 16 pages per field (c) 16 K bits per track (d) 64 fields
memory capacity	(a) 64 K words (b) 64 pages	(a) 1024 K words (b) 1024 pages
address register	word address: 16 bits	(a) field address: 6 bits (b) sector address: 4 bits (c) page address: 22 bits (d) word address: 10 bits

* there are 36 read/write heads per field
** K represents a multiple of 1024

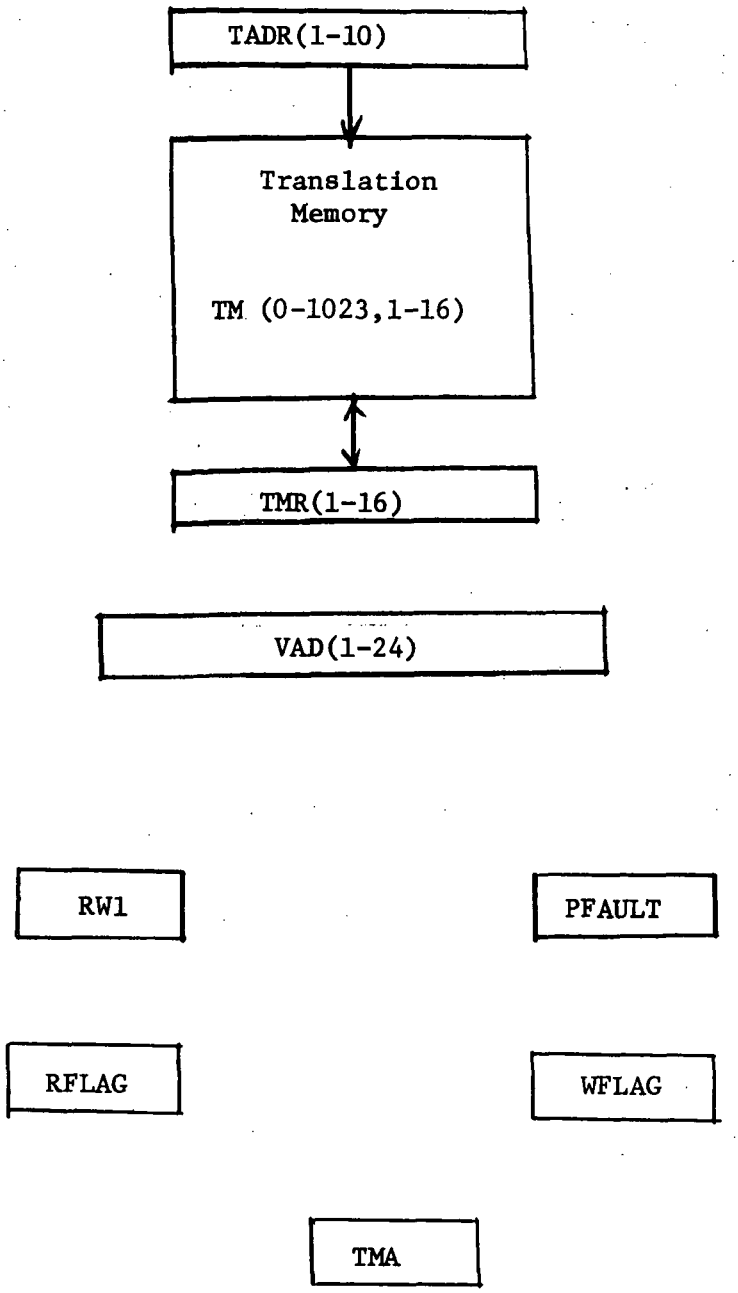
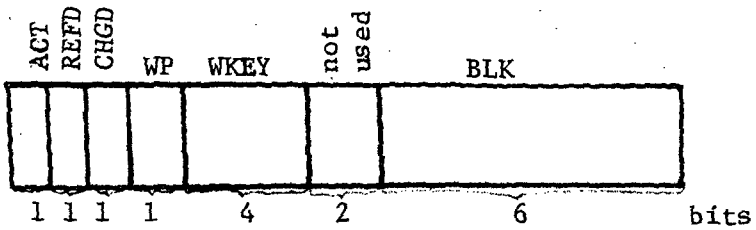


Fig. 3 Configuration of the Translation Memory



ACT=1, page is active in the main memory.

REFD=1, page has been referenced

CHGD=1, page has been written into the main memory

WP=1, page is write protected

WKEY, protection key of the page

BLK main memory page address if the page is in the main memory

Fig. 4 Translation memory word format

The 1-bit ACT field, when 1, means that the page is active in the main memory. The 1-bit REFD field, when 1, means that the page has been referenced. The 1-bit CHGD field, when 1, indicates that the page has been written into the main memory. The 1-bit WP field, when 1, means that the page is write-protected. The 4-bit WKEY field contains the protection key of the page. Lastly, the 6-bit BLK field contains the main memory page address corresponding to the virtual page if it is in the main memory.

The configuration of the translation memory is now described by the following CDL statements.

```
Comment, translation memory

Memory, TM(TADR)=TM(0-1024,1-16)   $translation memory

Register,     TADR(0-8),     $translation memory address register

              TMR(1-16),     $translation memory buffer register

              VAD(1-24),     $virtual address register

              TMA,           $translation memory access register

              RW1,           $read/write register (for CPU)

              RFLAG,         $read error flag

              WFLAG,         $write error flag

              PFAULT,        $page fault register

Subregister,  TMR(ACT,REFD,CHGD,WP,WKEY,NOTUSED,BLK)=TMR(1,2,3,4,5-8,9-10,11-16)
              VAD(D,I,X,PAGE,WORD)=VAD(1,2,3-4,5-14,15-24),
```

The translation memory contains a translation table which accepts a 10-bit virtual memory page address and delivers a 6-bit main memory page address if the page is in the main memory. This translation table is formed by the operating system for the program in execution (see p. 11 in reference 7).

The operation of translation memory TM is described by the sequence chart of Fig. 5. A read or a write sequence is started when register TMA contains a 1. Register RW1 is next tested for a read or a write operation. If RW1 is 0, it is the write sequence. The translation memory address register TADR is set to the given address and the translation memory buffer register TMR receives the input data; then the input word is transferred into the translation memory from the buffer register TMR. If RW1 is 1, it is the read sequence. The translation memory address register is set to the given address and the desired word is transferred from the translation memory into the buffer register TMR. Lastly, regsiter TMA is set to 0 and the TM returns to the waiting loop.

2.4 Page table memory

In order that the CPU can keep track of main memory page assignments (e.g., available pages), and in order that the paging drum channel can keep a list of pages to be swapped, a fast page table memory (PTM) is provided. This memory has a capacity of 64 66-bit words. It stores a page table. Each entry of the page table (that is, each word of the page table memory) stores a page descriptor. Thus, there are 64 page descriptors. Each page descriptor provides for each main memory page the following information:

(a) the current status of the page,

(b) the task which the page is or was attached to,

(c) the protection bits,

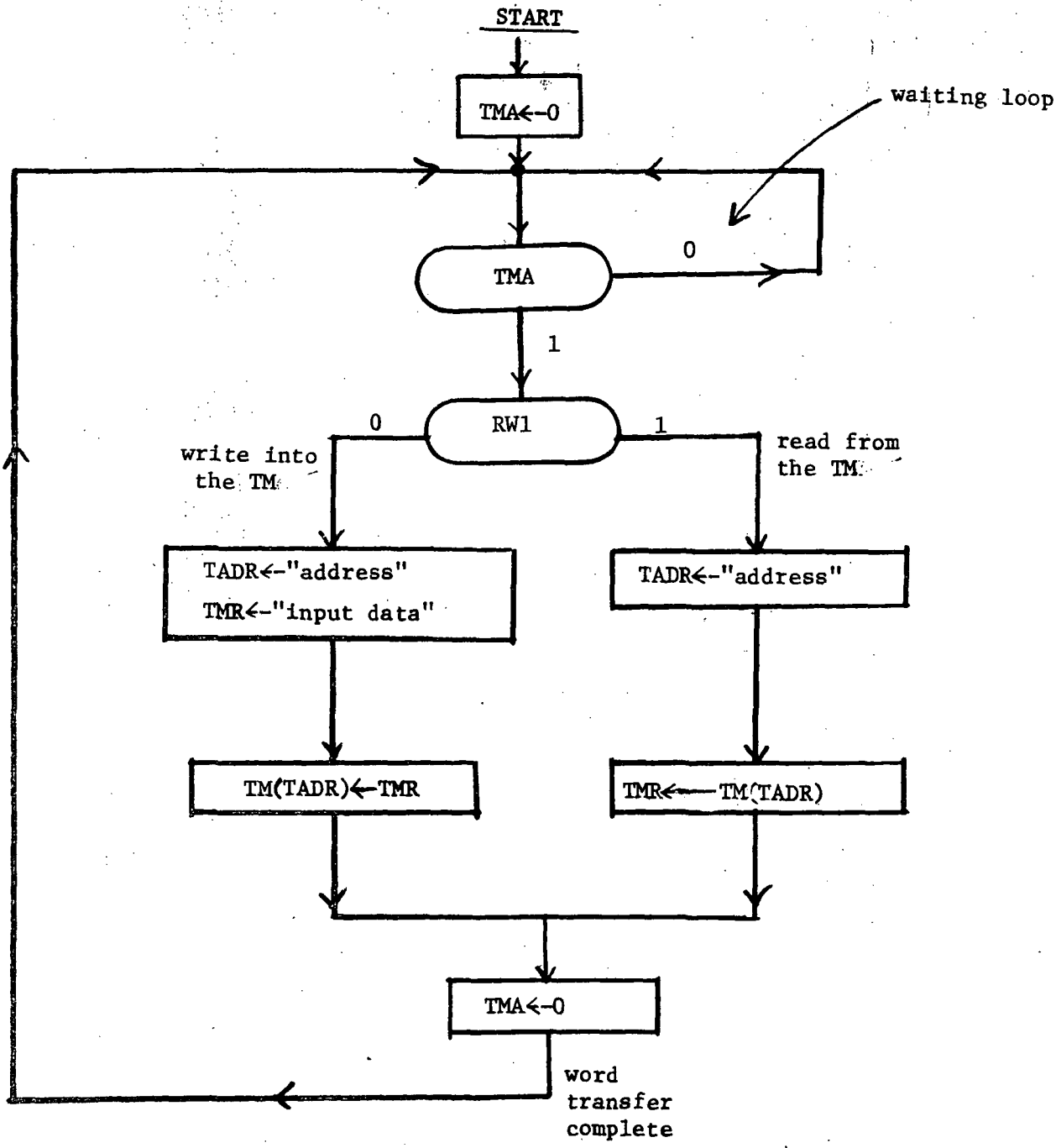


Fig. 5 Sequence chart for the translation memory

(d) utilization information,

(e) the corresponding virtual address of the page,

(f) the drum address of the page, and

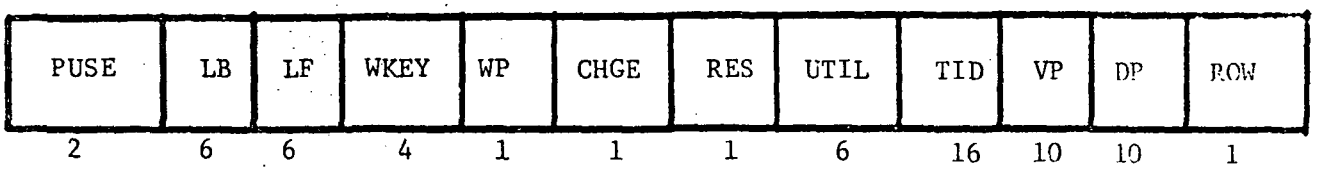
(g) list linkage information.

By means of the linkage fields of these descriptors, the 64 pages in the main memory can be linked together into one or more lists such as available page list and swappable page list.

The page descriptor format is shown in Fig. 6. There are 12 fields in a page descriptor. However, only fields LB, LF, DP, and ROW are needed for the simulation here. LB is the backward link pointing to the previous page descriptor of a list of page descriptors. LF is the forward link pointing to the next page descriptor of the list. DP is a 10-bit drum page address; the first 6 bits of DP specifies the field address of the drum page, while the other 4 bits of DP specifies the sector address. ROW is a read/write indicator. If ROW is 0, it is the read operation; else, the write operation.

2.5 Paging drum memory

The paging drum memory is a large rotating magnetic drum with a fixed head per track. The drum circumference is divided into equal parts called sectors, and the drum length is divided into groups of 36 tracks called fields. There are 16 sectors and 64 fields. The intersected area of a sector and a field is a drum page. There are 1,024 drum pages. A group of 36 bits in a field parallel with the axis form a drum word. There are 1,024 drum words in each drum page. Therefore, the drum page and the main-memory page are of the same size. The data transfer between the drum memory and the main memory is 36 bits at a time and one page at one transfer. The characteristics of the paging drum memory is also shown in Table 1.



LB: backward link

LF: forward link

DP: drum address of this page

ROW: read/write indicator

Note. The other fields are not used in the simulation. See reference (6).

Fig. 6 Page descriptor format

2.6 Paging Drum Channel

The paging drum channel PDC is essentially a dedicated processor that controls the operation of the paging drum in response to the paging-in and paging-out requests. A page of words is transferred from the drum through the PDC to the main memory when the CPU encounters a missing page (paging-in a page), or a page of words is transferred from the main memory through the PDC to the drum when the CPU releases a page (paging-out a page).

The configuration of the PDC is shown in the block diagram of Fig. 7. There are two memories in the PDC. Memory COM with address register SEC and buffer register COMMAND has a capacity of 16 52-bit words for storing 16 channel command words. Memory LISTS with address register SECTORS and buffer register PTL has a capacity of 16 12-bit words for storing 16 pairs of listheads. These 16 pairs are for the 16 sector lists in the page table memory.

The channel command word CCW format is shown in Fig. 8. There are five fields: the 1-bit C field, the 1-bit RWC field, the 8-bit CHAN field, the 6-bit PGE field, and the 36-bit FIRSTWORD field. When C is 0, there is no page transfer between the PDRUM and the MM. When RWC is 0, a drum page is to be transferred from the PDRUM to the MM through the PDC. When RWC is 1, a MM page is to be transferred from the MM to the PDRUM through the PDC. Only 6 bits of the 8-bit CHAN field are used for addressing the 64 fields in a drum sector. The 6-bit PGE field contains the MM page address. The PGE field must be non-zero since we assume that MM page 0 is not available. The 36-bit FIRSTWORD field contains the first actual word of MM page just in case the transfer is from the MM to the PDRUM.

The listhead format is shown in Fig. 9. There are two 6-bit fields. The first 6 bits specify the location of the front node and the last 6 bits specify the location of the rear node of the doubly linked sector queue in the page table memory.

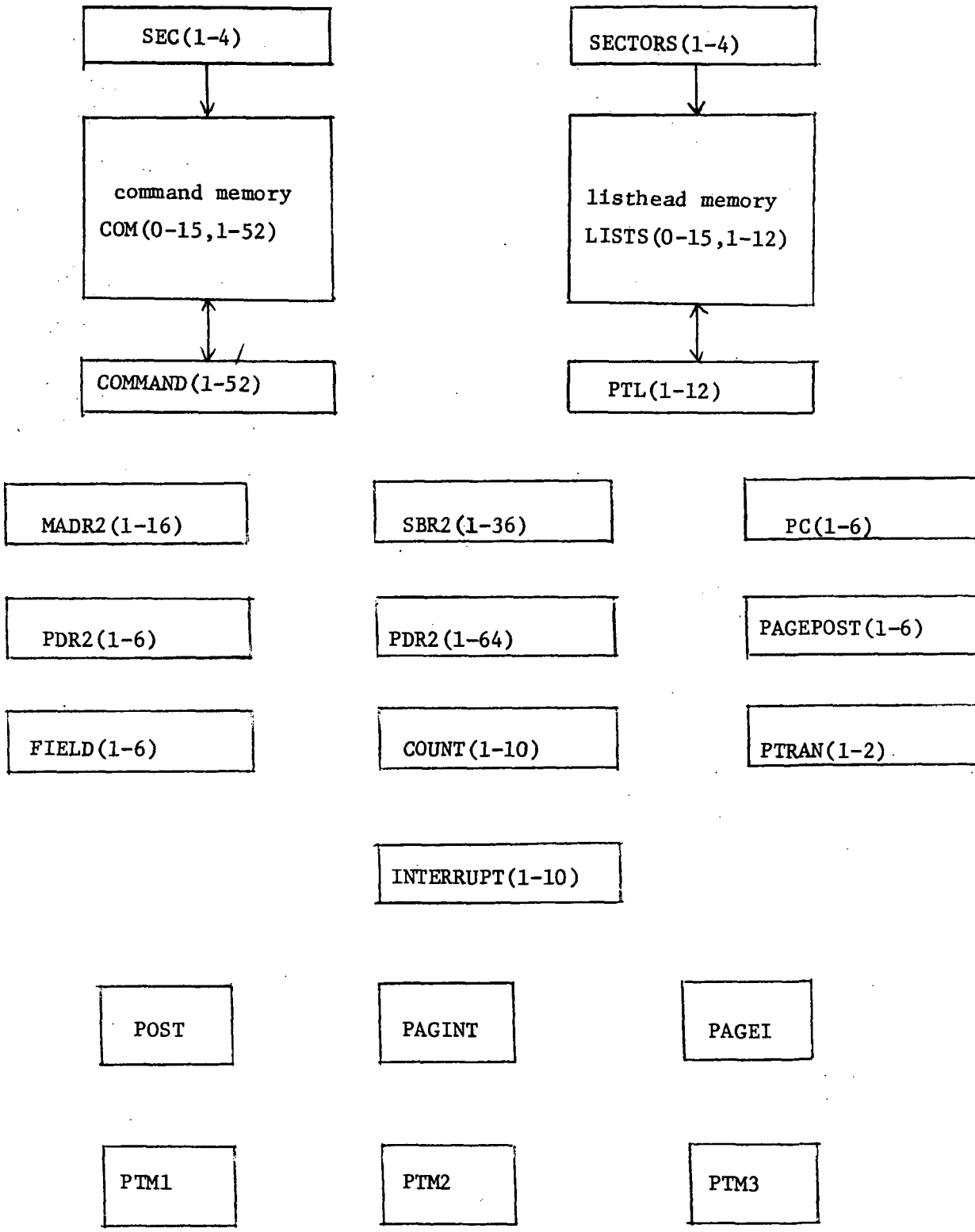
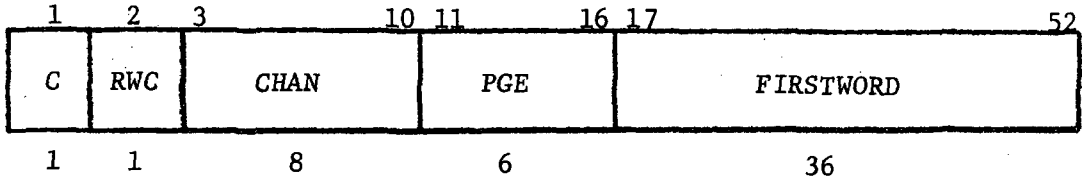


Fig. 7 Configuration of the Paging Drum Channel



C: no page transfer when C=0; else, there is a page transfer.

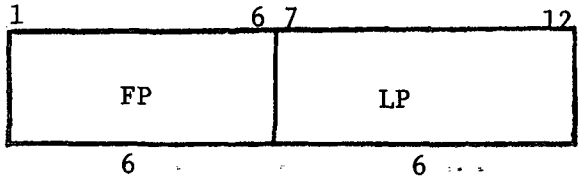
RWC: page to be read when 0; else, page to be written.

CHAN: drum field address

PGE: MM page address

FIRSTWORD: first word of the transferring page

Fig. 8 Channel Command Word Format



FP: the front listhead of the doubly linked list in the page table memory for use when an element of the list is detached.

LP: the rear listhead of the doubly linked list for use when an element of the list is inserted.

Fig. 9 Listhead format

3. Architecture of the virtual memory

This section presents the architectural design of the virtual memory. It describes the address translation table, various lists, the channel command words, the listhead table, the handling of the paging requests, and finally, various functions required for the virtual memory.

3.1 Address translation table

The translation of the virtual page address into the physical main memory page address is achieved by the address translation table in the translation memory. The basic idea is simple. It is a table with two columns: the left column with the virtual page address and the right column with the main memory page address. However, the left column is not required when a random-access memory is used, as it can be served by the address register of the random-access memory. As shown previously, several bits are added to each translation memory word to store control and status information required for each main memory page.

3.2 Lists

There are 64 pages in the main memory. Each page is pointed by a page descriptor stored in the page table memory. The pagetable memory address of each page descriptor corresponds to one main memory page address. Thus, 64 page table memory words store 64 page descriptors for 64 main memory pages. The format of the page descriptor has been shown in Fig. 6. As shown, there are two 6-bit fields LB and LF (backward link and fordward link) in the page descriptor. By means of these backward and forward links, the 64 main memory pages can be linked into one or more lists in the page table memory. Each node of the list is doubly linked with one link pointing in the forward

direction by the forward links LF's and the other link pointing in the backward direction by the backward links LB's.

There are many lists of page descriptors linked by fields LB and LF of the page descriptors as follows:

(a) one available-page list which links those pages of the MM that are available to the CPU;

(b) one swappable-page list which links those pages in the MM that are released by the CPU to be transferred from the MM to the drum;

(c) one or more user's lists, each of which links those pages of the MM that belong to a particular user;

(d) sixteen sector lists, each of which links those pages of the MM that are waiting (i.e., already in the queue) to be transferred either from the MM to the drum or vice versa. Since these 16 lists are used as queues, they are also called sector queues.

To enable a quick access of the first entry and the last entry in each sector list, two 6-bit listheads are provided for each sector list. The 32 listheads for the 16 sector lists are stored in the 16 words of the LISTS memory. The listheads for the available-page list and the swappable-page list are stored in register LAVP(1-12) and LSP(1-12), respectively. The listheads for the current user is stored in register PTLIST(1-12) or register GPTL(1-12); the listheads for the users' lists are stored in the system table permanently resident in the main memory described elsewhere (6).

3.3 Channel Command Words

The words in the COM memory are called Channel Command Words or CCW's. Each CCW stores the following pertinent information for initiating and controlling a page transfer:

(a) main memory page address,

(b) drum field address,

(c) read/write operation,

(d) transfer request, and

(e) the first word of the main memory page in case the transfer is from the main memory to the paging drum.

There are 16 CCW's; each CCW contains control information for handling its drum sector. There is a linkage between each page descriptor and the channel command word constructed from this page descriptor. This linkage is the main memory page address which is stored in the channel command just constructed. This main memory page address is used as the page memory address, by means of which the page descriptor in the page table memory can be located.

There is an important exception to the way that the first page descriptor of every sector list is linked: this first page descriptor is detached from its sector list, after its pertinent information for initiating the data transfer is used to construct the CCW for that sector. The page descriptor of this page can be located in the page table memory by the CCW whose field PGE holds the MM page address of this page. The reason for not linking the first MM page to its sector list is to enable the immediate accessability of this page information from the COM memory in the PDC, since the COM memory is exclusively accessable by the PDC while the page table memory is accessable by the PDC and the CPU. As a result, the PDC can rapidly respond to the drum each time when a new drum sector begins to be scanned. In other words, as the drum heads reach the beginning of each sector, the CCW of this sector is accessed from the COM memory and the data transfer, if called for, is initiated right away. After the initiation, the current

CCW is of no further use; this COM memory location can now be refilled with the pertinent information for initiating the next page transfer for the same sector which occurs when the drum completes another revolution and again begins to scan this sector. This refilling is accomplished as follows. While a page is being transferred to or from a drum sector, the next page descriptor is detached from the sector list in the page table memory and the pertinent information of the page obtained from the page descriptor is used to construct a CCW for the current drum sector. This CCW will be used after one drum revolution.

There are 16 channel command words in the COM memory. Since the first page descriptor of each sector list is not linked to that sector list, there can be as many as 16 main memory pages that are not linked at all by the page descriptors. However, the page descriptors of these main memory pages are pointed by (and thus indirectly linked by) the channel command words as described previously.

3.4 Listhead table

For each drum sector, the paging drum channel needs to quickly locate the corresponding sector list for converting the first page descriptor on that sector list into a channel command word. The pointers or listheads for locating these 16 sector lists are the entries of the listhead table stored in the LISTS memory.

There are 16 entries in the listhead table; each entry occupies one LISTS memory word. As shown previously in Fig. 9, each LISTS memory word contains a pair of listheads which point to the front and the back of the corresponding sector queue in the page memory. By means of the pair of the listheads, the paging drum channel can quickly locate the sector queue for

the drum sector now being scanned by the drum heads.

3.5 Page-transfer requests

As mentioned previously, the swappable-page list links those pages in the main memory that are released by the CPU to be transferred to the drum, while each of the sector lists links those main-memory pages queued to be paged-in or paged-out. The reason why two kinds of lists are required is that the operating system of the computer system aims to schedule alternately the read and write page transfers in the sector lists.

Fig. 10 is a diagram showing the queuing of page-transfer requests in the sector queues. The 16 sector lists are maintained by the PDC; the available-page list and the swappable-page list are maintained by the CPU. When a page is requested by the CPU and found missing in the main memory, a page-fault interrupt is generated; this interrupt signifies that a new page is to be paged-in. The CPU allocates a page descriptor from the available-page list and posts a read page-transfer request to the PDC. The PDC responds by placing the request in the appropriate sector list for the drum sector where the page is stored. The CPU next posts a write page-transfer request to the PDC for swapping out a page from the swapping-page list. In this manner, the read and write page transfers are scheduled alternately, as also indicated in Fig. 10.

Fig. 11 is a diagram showing the handling of the lists. As mentioned before, the CPU posts a read page-transfer request owing to a page fault or posts a write page-transfer request owing to a page release. These requests are entered by the CPU into the appropriate sector lists. The PDC next generates channel command words for each drum sector from the information in the sector lists and the listhead table. Whenever a page transfer is completed, the PDC signals the CPU. The CPU then updates the user's lists and resumes the execu-

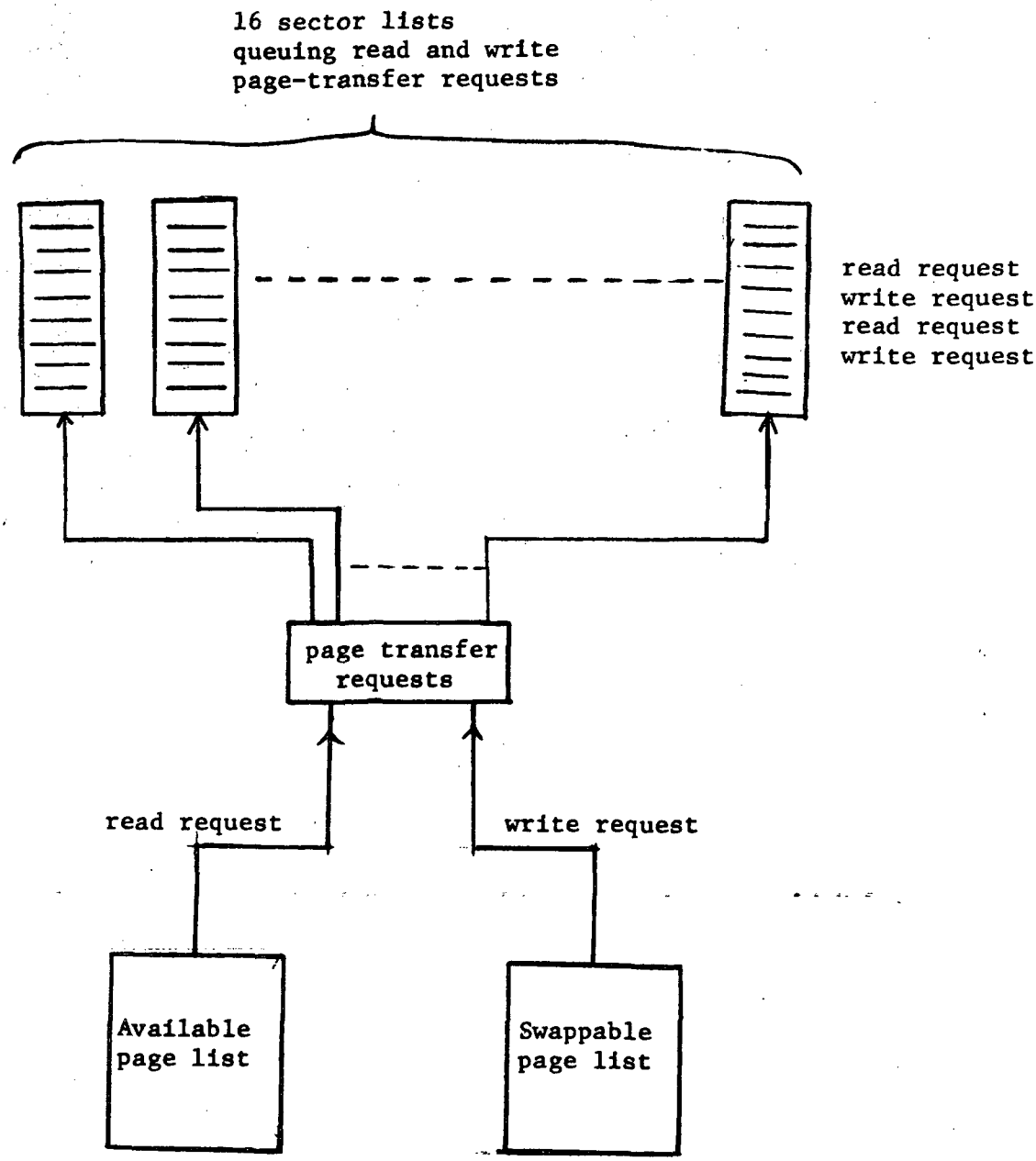


Fig. 10, Quening of paging-in (read) requests
and paging-out (write) requests

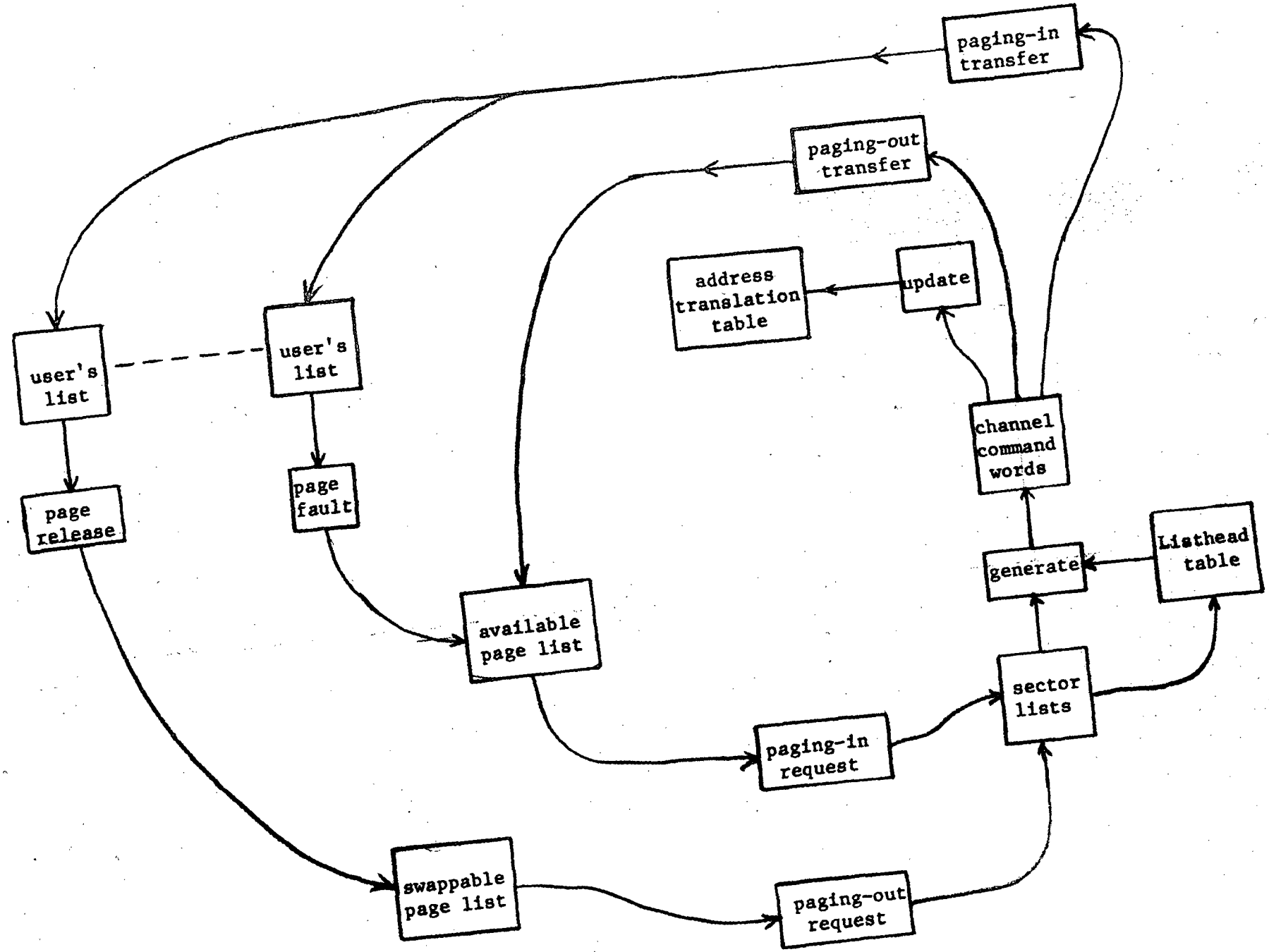


Fig. 11, Handling of the lists and tables

tion of program, or it links the page descriptor now released to the available page list. The CPU posts page-transfers one after another, while the PDC initiates and executes the page transfers in an alternate read and write operation in order to optimize the drum transfer operation. The CPU handles all the lists including the users' lists except the sector lists which is handled by the PDC.

3.6 Architectural design

The architectural design of the virtual memory is presented in the diagram of Fig. 12. In this diagram, major functions of the virtual memory are indicated: virtual address translation, generation of page descriptors, queueing of paging requests, generation of PDC command words, and execution of page transfers, in addition to functions handled by the operating system (not shown).

The first function is virtual address translation. The CPU examines the virtual address of an instruction, and determines from the translation memory whether the page specified by the virtual address is in the main memory. If it is in the main memory, the virtual address is translated into a main memory address by address translation table in the translation memory. The particular main memory address derived from the virtual address is then used to fetch the desired main memory word. If this page is not in the main memory, then a page fault is generated to signal an interrupt. The operating system then takes over; it examines the cause of the interrupt and issues a paging-in request.

The second function is generation of page descriptor. When a paging-in request is issued, a page descriptor is generated by the CPU, using the virtual address, the main memory address, and the required transfer operation. This page descriptor contains the page address in the drum, the corresponding

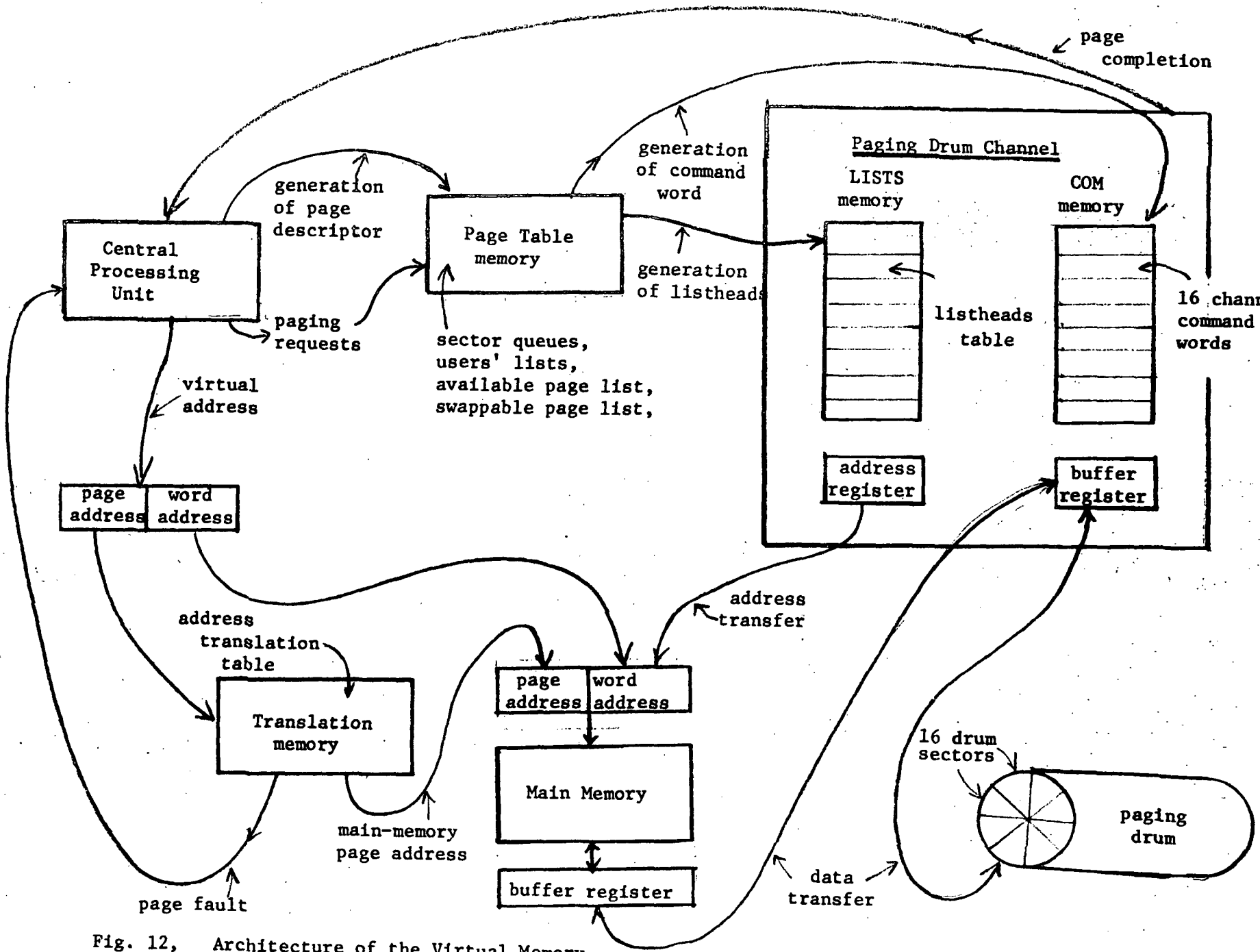


Fig. 12, Architecture of the Virtual Memory

virtual address of the main page, and other control information provided by the operating system, as mentioned previously. This page descriptor is inserted and linked into one of the 16 sector queues.

The third function is the queuing of paging requests. As mentioned, there are 16 sector queues in the page table memory. Each sector queue stores the paging-in and paging-out requests for one drum sector. The requests are processed by the paging drum channel at an optimum time later.

The fourth function is execution of page transfers. During each drum revolution, the drum heads scan 16 sectors. At the begining of scanning each sector, the channel command word for that sector is taken out of the COM memory, and the page transfer specified by this channel command word is carried out. During each drum revolution, there may be as many as 16 page transfers.

The fifth function is generation of channel command words. As mentioned, each channel command word in the COM memory is actually the first paging request of a sector queue in the page table memory. While a channel command word is being executed for the drum sector that is being scanned by the drum heads, the first entry of the current sector queue is fetched from the page table memory and converted into the channel command word for the current drum sector to be executed during the next drum revolution.

There are a number of virtual memory functions that are carried out by the operating system: paging interrupt handling, drum page allocation, initial loading, and execution of virtual memory policies. Paging interrupts are those due to paging faults, paging requests and paging completions. Interrupts including these paging interrupts are handled by the interrupt routine of the operating system. Drum page allocation refers to the allocation of available pages on the drum to a user's program. Initial loading refers to the loading of the program into the allocated drum pages. Drum storage alloca-

tion and initial loading are dynamic storage allocation functions of the operating system. Virtual memory policies are to be further discussed; execution of the replacement policy is assigned to the operating system.

4. Operation of the Virtual Memory

This section presents a flow chart which shows an overview of the operation of the virtual memory. It then describes more fully the virtual address translation, the paging policies, and the program for initiating and controlling page transfers.

4.1 An overview

Fig. 13 is a flow chart showing an overview of the virtual memory operation. As shown, the virtual address is first fetched by the CPU. By means of the address translation table in the translation memory, whether the page addressed the virtual address is in the main memory or not is determined. If the page is in the main memory, the CPU continues to execute the current instruction. If it is not in the main memory, execution of the current program is suspended and a page fault is issued. The operating system identifies the cause of interrupt and then examines the available-page list. If no page is available as indicated from the available-page list, the operating system next examines the swappable-page list.

If the swappable-page list indicates that no page is available for swapping, the operating system selects a page to be paged out by applying the replacement policy. Once a main memory page is selected to be paged out, its address is inserted into the swappable-page list. The swappable-page list is no longer empty, and it now becomes the case that a page is available for swapping. The operating system issues a paging-out request and calls the

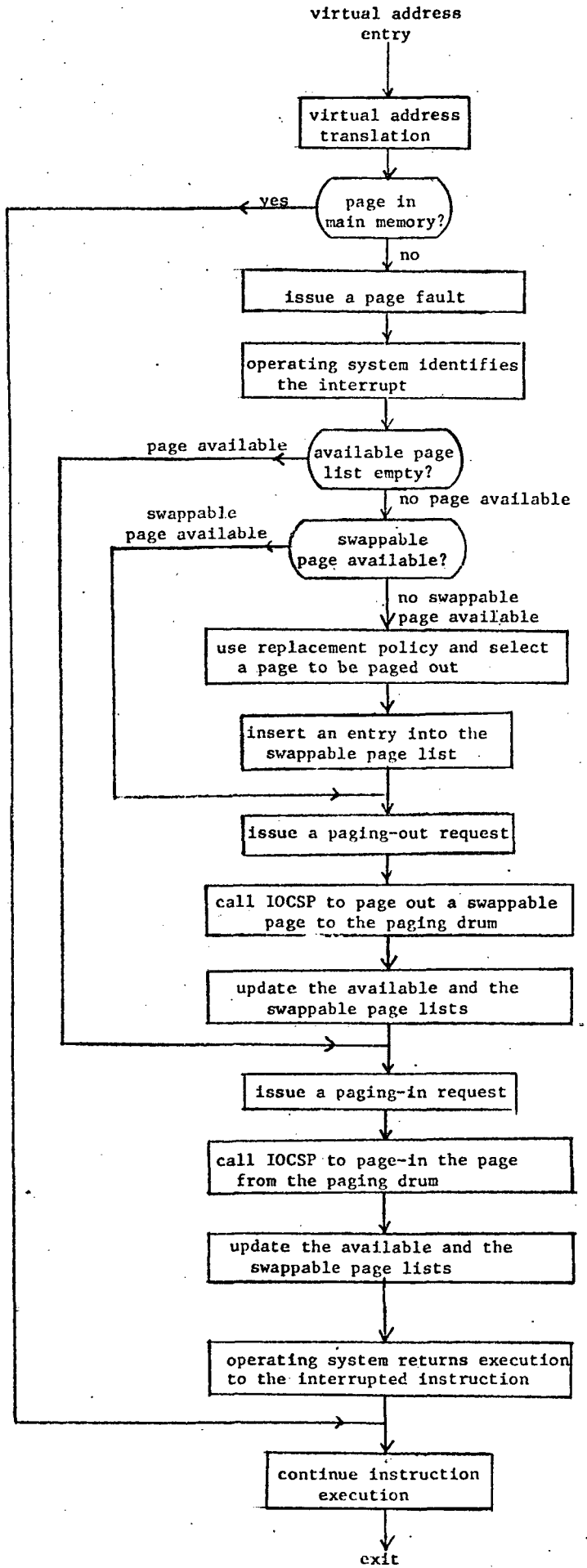


Fig. 13, Flow chart showing an overview of the virtual memory operation

IOCSP to page out the swappable page to the paging drum, and then updates both the available page list and the swappable page list. Now, the available-page list is no longer empty. It becomes the case that a page is available for paging-in.

When a page is available, the operating system issues a paging-in request, calls the IOCSP to page in the required page from the paging drum, and then updates the available page list and the swappable-page list. The operating system now returns to the executions of the interrupted instruction.

The operation shown in Fig. 13 could be and should be improved. First, the page swapping could be made concurrent with other paging operations so that the available page list will be built up to make pages available for allocation. Second, after the desired page is paged in, the operating system does not have to return immediately to the interrupted instruction, depending on the scheduling algorithm that is adopted by the operating system.

4.2 Virtual address translation

Fig. 14 shows the configuration for virtual address translation. The translation begins with a virtual address in the 24-bit virtual address register VAD and ends with the physical main memory address in the 16-bit main memory address register MADR1(1-16). Register MADR1 consists of two parts: subregister MADR1(1-6) which contains the main memory page address and subregister MADR1(7-16) which contains the word address of the page. As shown in Fig. 6, subregister VAD(5-14) is connected to the translation memory address register TADR; subregister VAD(15-24) is connected to subregister MADR1(7-16); and subregister TMR(11-16) is connected to subregister MADR1 (1-6).

The algorithm for the virtual address translation is shown in the sequence chart of Fig. 15. It begins when the virtual address "P-W" is transferred from the CPU to register VAD. Since subregister VAD(5-14) contains the location of the translation memory word that has the main-memory page address, it is transferred to the translation memory address register TADR. The desired translation memory word is next read out of the translation memory into register TMR. Then, the contents of subregister TMR(11-16) are transferred to subregister MADR1(1-6), while the contents of subregister VAD(WORD) are transferred to subregister MADR1(7-16). Thus, register MADR1 now contains the main memory address.

4.3 Paging policies

There are three policies involved in the paging transfers: replacement policy, fetch policy and placement policy. The replacement policy determines which page in the main memory is to be paged out. The fetch policy decides when the page is to be paged in. The placement policy decides where

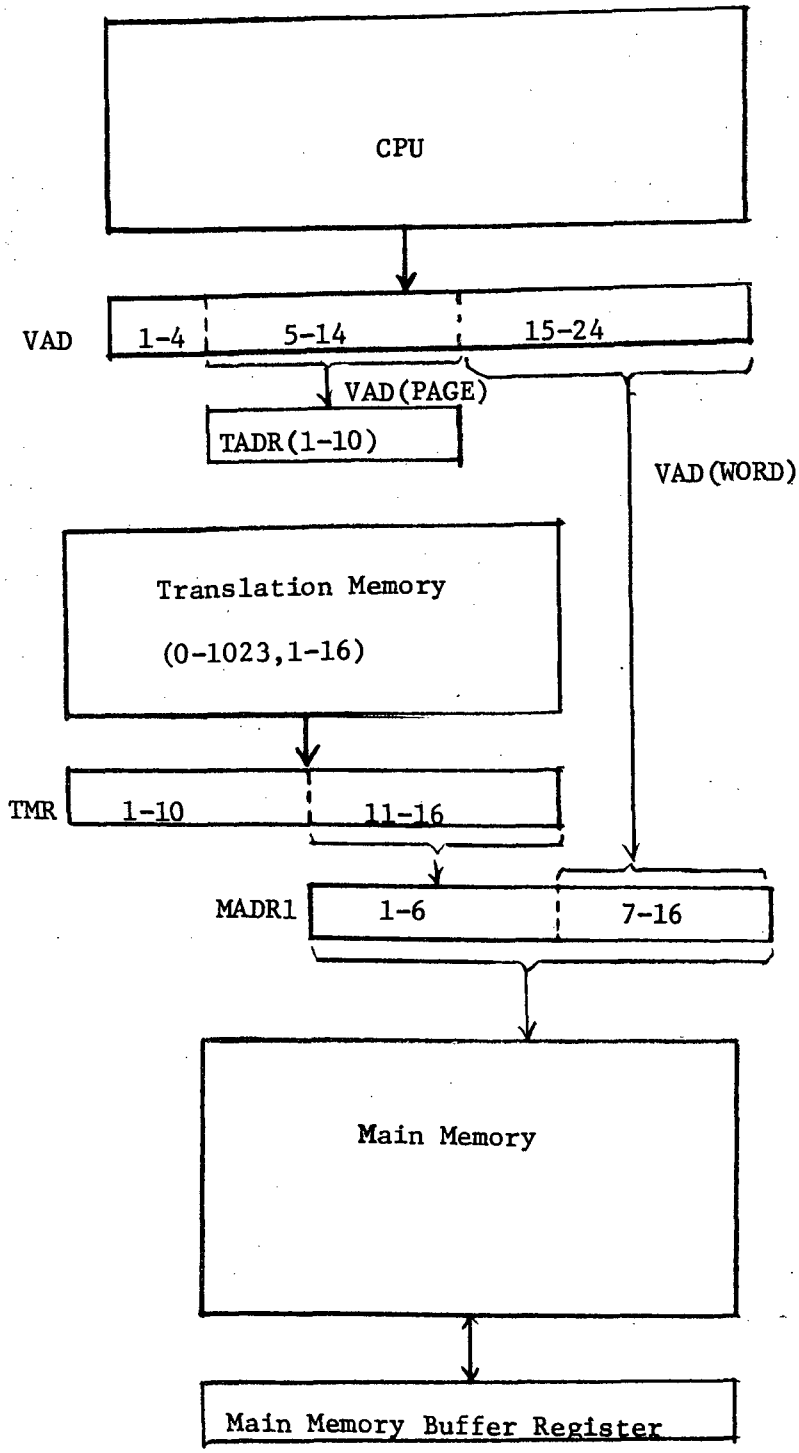


Fig. 14 Configuration for the virtual address translation

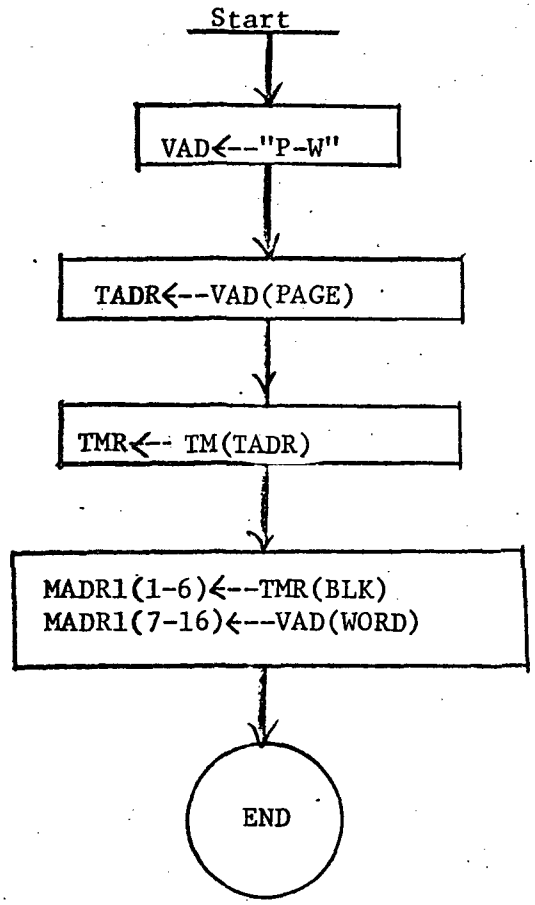


Fig. 15 Sequence chart of the virtual address translation

in the main memory the page is to be placed.

A commonly used paging policy is the demand paging. In demand paging, one page of the program is paged in at the beginning of the time quantum in a time-sharing system; each additional page is then paged in when the page is demanded (i.e., when a page fault occurs). At the end of the time quantum, no pages of the program are removed from the main memory; its removal is determined by the replacement policy. This design employs the demand paging as the fetch policy. The placement policy makes use of the previously discussed lists: the available page list and the swappable list. The entries in these two lists are made available by the replacement policy.

There are a number of known replacement policies:

(a) select the page randomly,

(b) select the page which has been in the main memory the longest time,

(c) select the page which is the least recently referenced by an active process,

(d) select a page from the working set of an inactive process or a page not in the working set of any active process.

The working set of a process is the smallest set of pages which must be in the main memory in order that the process may run efficiently. The replacement policy chosen in this design is the working set model as has been described elsewhere (6, 9, 10).

4.4 Input-output Control System for Paging

The Input-Output Control System for Paging (IOCSP) for the virtual memory is a program that initiates and controls automatic transmission of data to and from the paging drum memory. The IOCSP performs the following functions:

(a) it accepts page-transfer requests;

(b) it constructs paging request entries;

(c) it constructs channel programs for the paging drum channel;

(d) it initiates the PDC to execute the channel program;

(e) it updates the address translation table after a page-transfer operation is completed.

Fig. 16 is a flow chart which shows handling of paging requests by the IOCSP. As shown, after initialization, the IOCSP accepts paging requests from the CPU. It then constructs a paging request entry and places it in the paging-request queue. For each paging request, it constructs a paging-drum channel program and stores it in the main memory. This channel program is next transferred to the command memory of the PDC and is then executed by the PDC. After the execution is completed, the address translation table is updated. Finally, the CPU notified.

The above functions of the IOCSP are implemented by four routines: IOCS1, UNIT, FETCH, and IOCS2 to be further described.

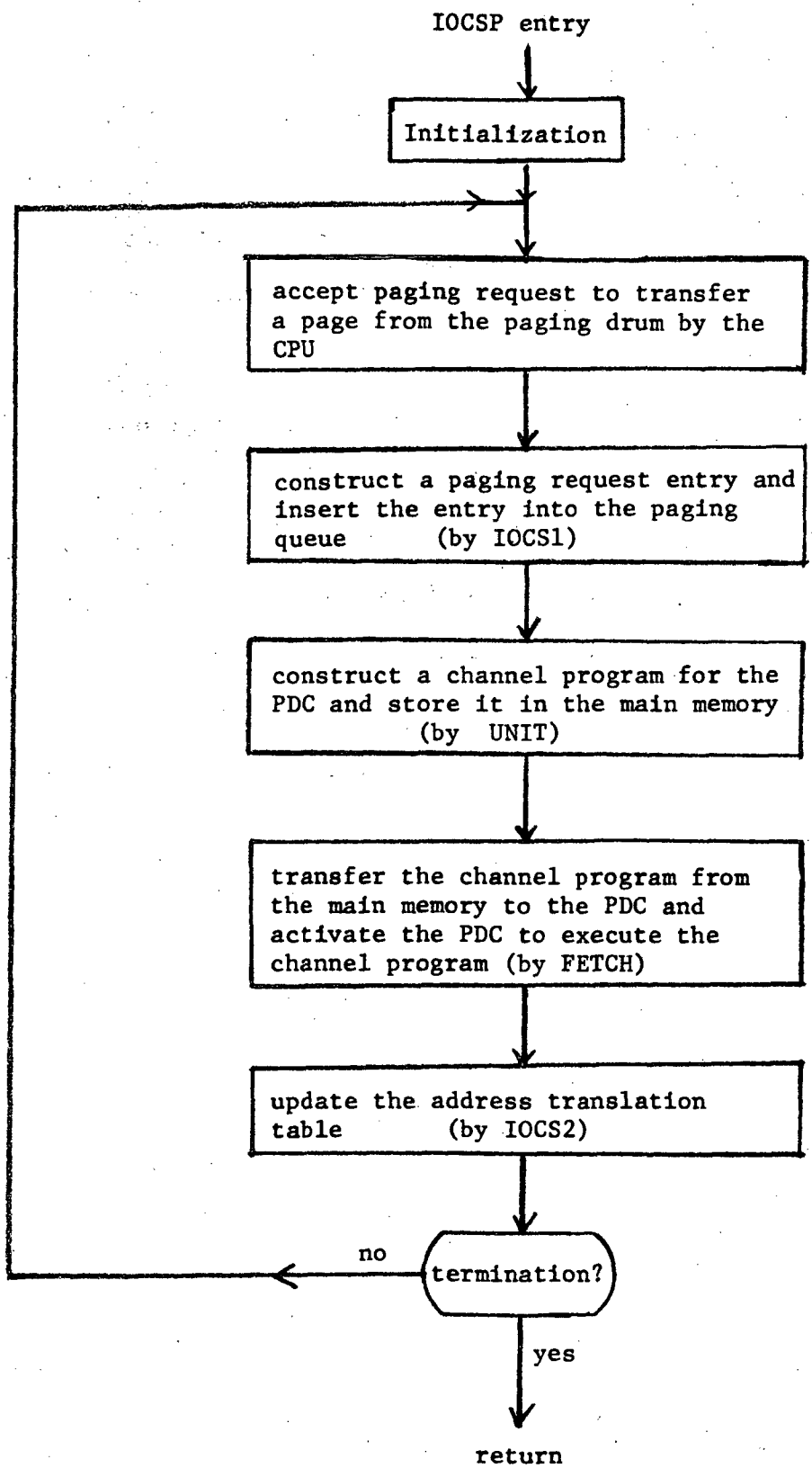


Fig. 16 Flow chart showing servicing a paging request by the IOCSP

5. The IOCSP Routines

The IOCSP consists of four routines:

(a) Paging Initiation Routine (IOCS1)

(b) Paging Drum Unit Interpretive Routine (UNIT)

(c) Fetch Routine (FETCH)

(d) Paging Completion Routine (IOCS2)

These routines are described below.

The Paging Initiation Routine is activated when a page-transfer is required during the execution of a user's program or the supervisor program. It determines the nature of the paging request (paging-in or paging-out), constructs a paging request entry described in the next section, and checks the availability of the paging drum.

The Paging Drum Unit Interpretive Routine is called by IOCS1. It checks the function code of the paging request (0 for paging-in and 1 for paging-out), generates a page descriptor and a channel command word to be executed by the PDC using the information in the given paging request entry, and calls the Fetch routine which is described below.

The Fetch Routine transfers the channel program from the main memory to the PDC, and then initiates the PDC to execute the channel program. Once the PDC is initiated, it carries out data transfer independent of the process in the CPU.

The Paging Completion Routine routine gains control when the page-transfer operation is completed. It marks the paging request entry as serviced, updates the address translation table, and then returns control to the user's program.

Detailed description of each of these routines is now given below.

5.1 Paging request queue

When the page addressed by the virtual page address of an instruction is in the main memory, the virtual address is translated into the main memory address and the instruction is then executed. If the page is not in the memory, this page has to be brought in from the paging drum memory by the IOCSP.

The IOCSP first places the paging request in the paging-request queue, which is a linear list of paging request entries in the main memory for the sake of convenience in later simulation. Each entry is called a node, representing one request. Table 2 shows the six fields of the node.

There are six fields in a node which represents a paging request. The mark field when 0 indicates that the request is not serviced; otherwise, it is serviced. The paging instruction field when 0 indicates the paging-in, otherwise, it indicates paging-out. Other fields give the MM page address, the virtual page address, the drum memory field address, and the drum memory sector address. Fig. 17 shows the (K-1)th node, the Kth node, and the (K+1)th node in the paging request queue.

5.2 Paging Initiation Routine (IOCS1)

The Paging Initiation Routine (IOCS1) gains control when there is a paging request to be serviced. The IOCS1 routine first constructs a paging request entry and then passes control over to the UNIT routine.

Fig. 18 is a flow chart of the IOCS1 routine. When IOCS1 routine starts, index K of the paging request entry is incremented by 1. Next, the value of K is examined. If K is greater than 64 which is the maximum paging request queue length, then the value of K is reset to 1 to avoid overflow. If K is less than 64, then a paging request entry is constructed as follows:

Table 2. Representation of a paging request entry by a 6 word NODE

Words of * the Kth node	Fields of the Kth node	Contents of words
NODE(K,1)	mark	0 for not serviced; 1 for serviced.
NODE(K,2)	paging instruction	0 for paging-in; 1 for paging-out.
NODE(K,3)	MM page address	MM page addresses range from 0 to 63
NODE(K,4)	virtual page address (same as translation memory address)	virtual page addresses or the translation memory addresses
NODE(K,5)	drum memory field address	drum field addresses
NODE(K,6)	drum memory sector address	drum sector addresses

* K is the index for the paging request

Note: The paging-in or paging-out operation, the MM page address, and the virtual page address are available from the operating system.

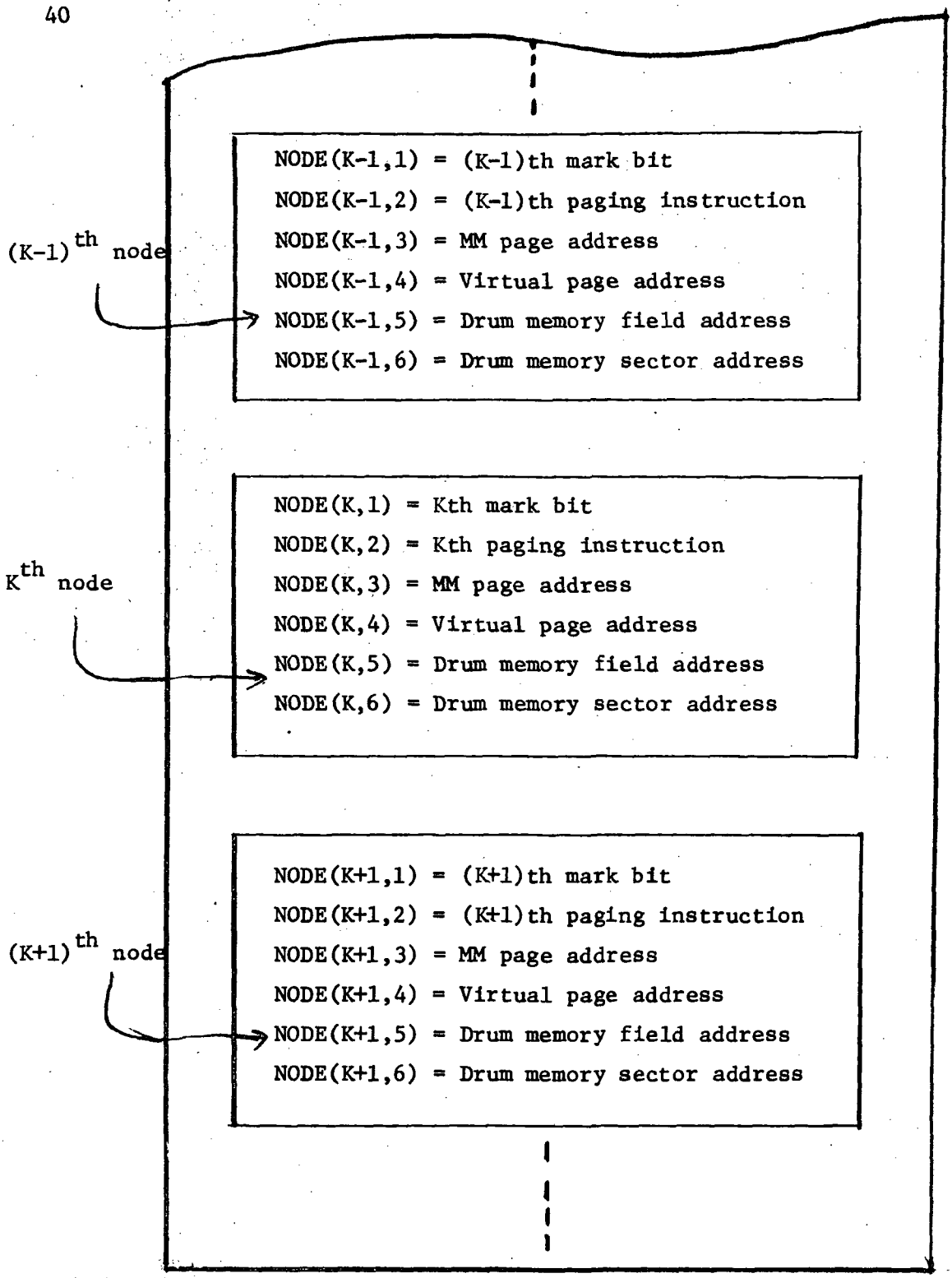


Fig. 17 The (K-1)th node, Kth node, and (K+1)th node in the paging request queue

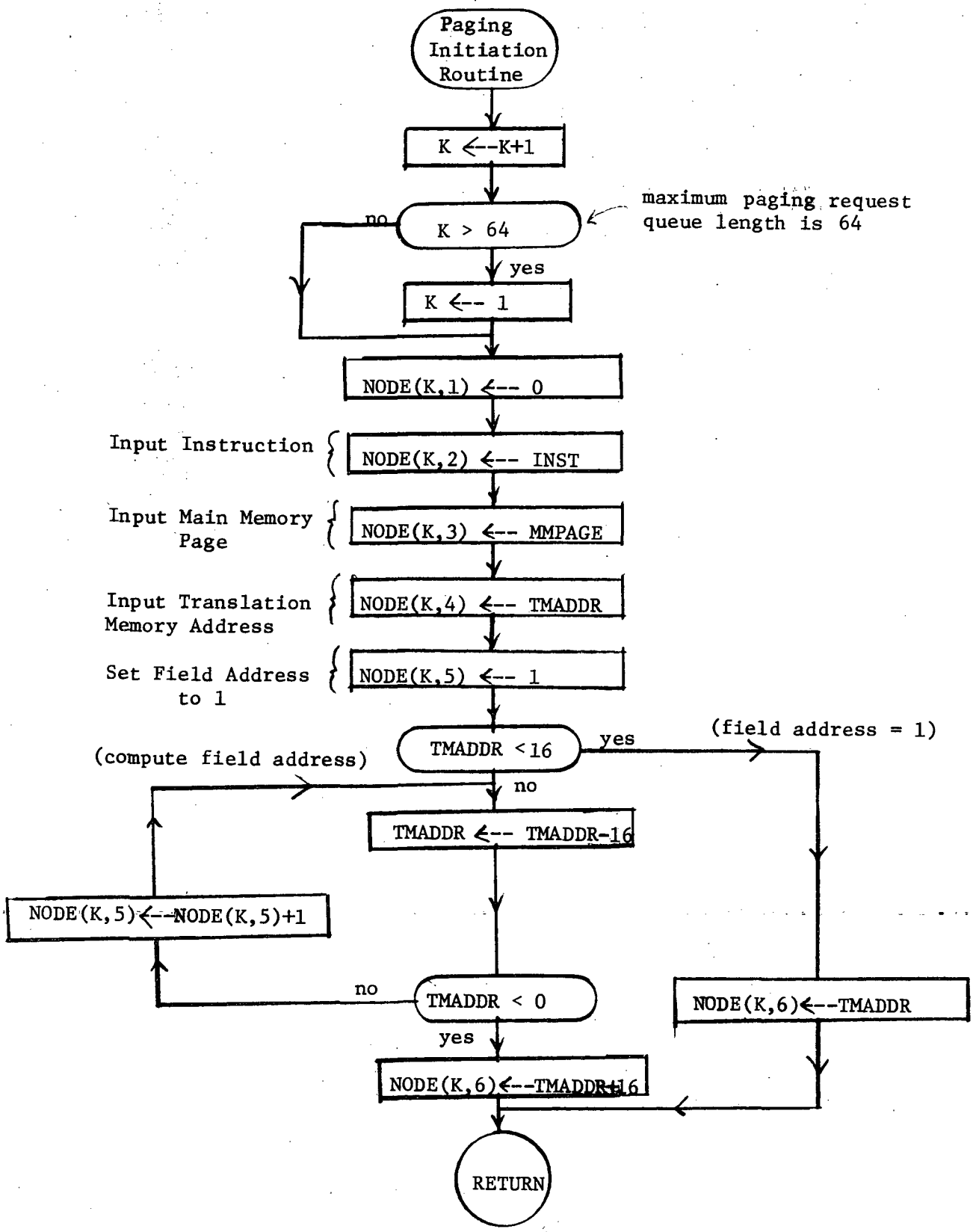


Fig. 18 Flow chart of the Paging Initiation (IOCS1) Routine

(a) NODE(K,1) is set to 0 since the paging request is not yet serviced.

(b) NODE(K,2) is set to 0 for paging-in or 1 for paging-out.

(c) NODE(K,3) is set to the main memory page address MMPAGE where 0 MMPAGE≤63.

(d) NODE(K,4) is set to the translation memory address TMADDR (same as the virtual page address) where 0≤TMADDR≤63.

(e) NODE(K,5) is set to the paging drum memory field address. Initially, it is set to 1.

(f) NODE(K,6) is set to the paging drum memory sector address.

Next, the drum memory field address and the drum memory sector address are computed. If TMADDR is less than 16, then drum memory field address is set to 1 and the drum memory sector address is the same as TMADDR. Therefore, NODE(K,6) is set to TMADDR. If TMADDR is greater than 16, then 16 is subtracted from TMADDR while 1 is added to NODE(K,5). The subtraction and addition operations continue until TMADDR becomes negative. When TMADDR is negative, then NODE(K+6) is set to (TMADDR+16) which is the computed drum memory sector address. After the paging request entry is constructed, the IOCS1 passes control to the UNIT routine. A simpler way is to shift the translation memory address 4 bit positions to the right.

5.3 Paging Drum Channel Unit Interpretive Routine (UNIT)

The Paging Drum Channel Unit Interpretive Routine (UNIT) is called by the IOCS1 after a paging request entry is constructed. The UNIT routine constructs a page descriptor for the main memory page to be transferred. It also constructs a channel command word for execution by the PDC.

A flow chart of the UNIT routine is shown in Fig. 19. When the UNIT routine starts, a CCW is constructed with the information in the paging request entry. The left half of the CCW is constructed according to NODE(K,2), NODE(K,3), and NODE(K,5) and the result is stored in the MM location 4. The **

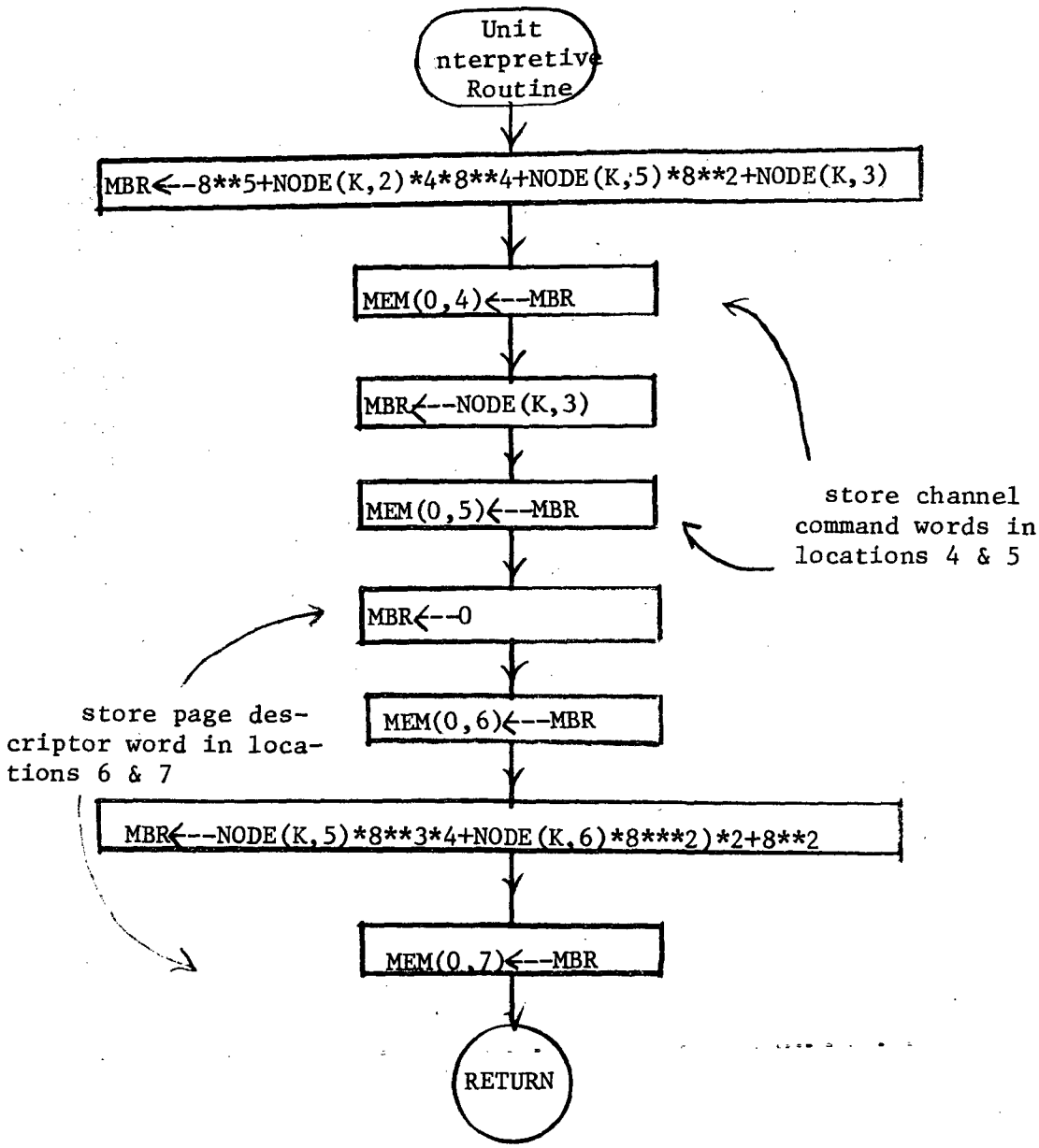


Fig. 19 Flow chart of the Unit Interpretive (UNIT) Routine

symbol means exponentiation which is used to shift control information in a computer word to the desired bit position. The right half of the CCW is set to NODE(K,3) and is stored in the MM location 5. Next, a page descriptor is constructed. The left half of the page descriptor is set to 0 and the result is transferred to the MM location 6. The right half of the page descriptor is constructed according to NODE(K,5) and NODE(K,6) and the result is stored in the MM location 7. After the constructed channel program is stored in the MM locations 4 through 7, the UNIT routine passes control to the Fetch routine.

5.4 Fetch Routine (FETCH)

The Fetch Routine (FETCH) transfers control information from the main memory to both the page table memory and the PDC COM memory.

The flow chart showing the FETCH routine is shown in Fig. 20. When the FETCH routine begins, the contents of the MM location 4 are transferred to the left half of buffer register COMMAND of the COM memory in the PDC, and the contents of the MM location 5 are transferred to the right half of register COMMAND. Address register SEC of the COM memory is next set to NODE(K,6) in order that the CCW for the page-transfer operation can be transferred into the proper location of the COM memory.

Next, the contents of the MM location 6 are transferred to the left half of the page table memory buffer register PTR, and the contents of the MM location 7 are transferred to the right half of register PTR. The page table memory address register PADR is set to the contents of NODE(K,3) in order that the page descriptor can be stored into the proper location of the page table memory. At the end, the FETCH routine initiates the PDC to execute the channel program.

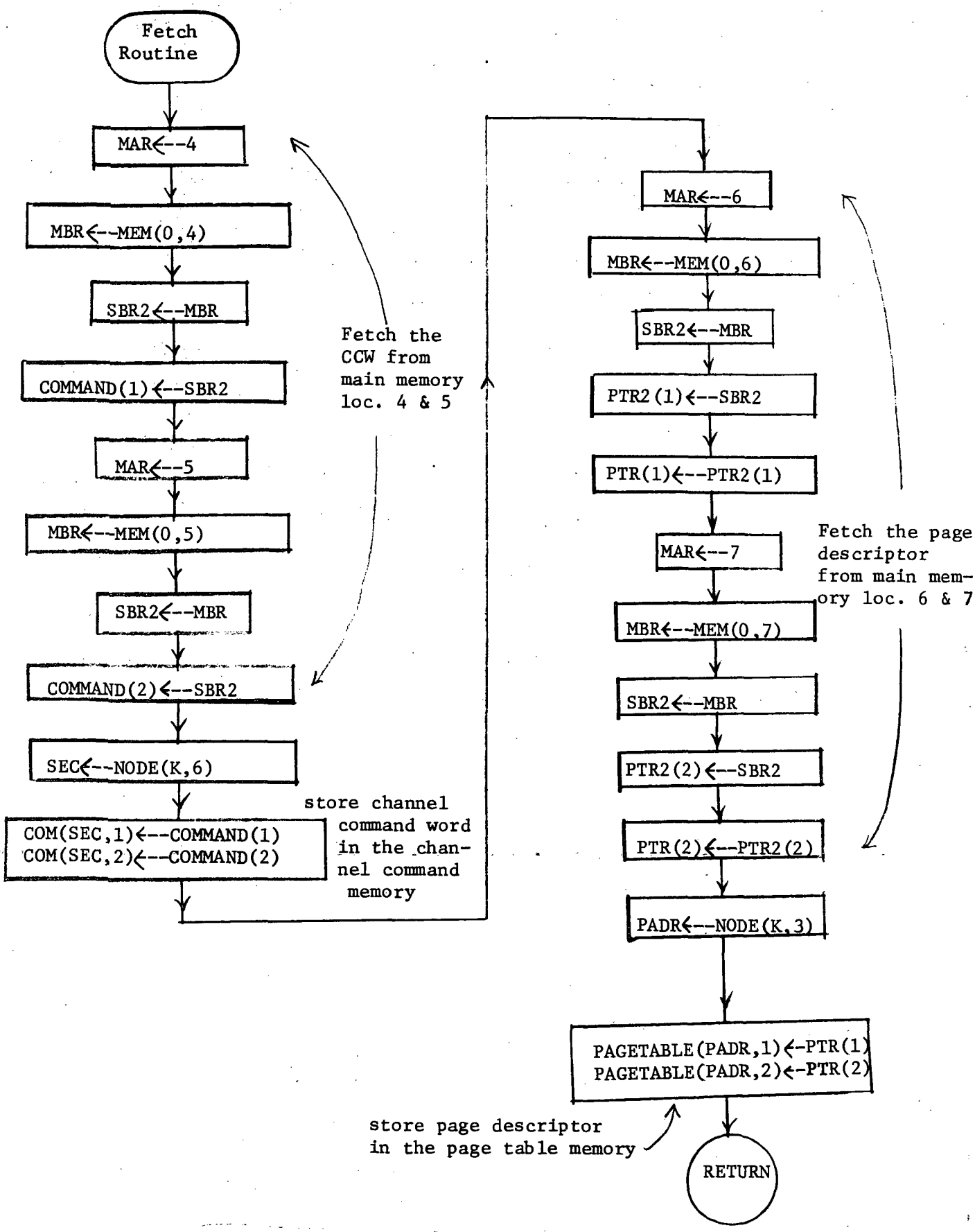


Fig. 20 Flow chart of the Fetch (FETCH) Routine which transfers the channel program from the main memory to the channel

5.5 Paging Completion Routine (IOCS2)

The paging completion routine IOCS2 gains control when a bit is set in the interrupt register INTERRUPT. The flow chart showing the IOCS2 routine is shown in Fig. 21. When the IOCS2 routine starts, the paging request entry is marked as serviced by setting the mark field NODE(K,1) to 1. Next, the translation memory address register TADR is set to NODE(K,4) and the corresponding translation memory word is transferred to the translation memory buffer register TMR. NODE(K,2) is then tested to see if a paging-in or a paging-out request was serviced.

If NODE(K,2) is 1, a paging-out operation has been completed. The contents of register TMR are modified such that the "in-core bit" is 0. If NODE(K,2) is 0, completion of a pagin-in operation is indicated. The content of register TMR are modified such that the "active bit" is set to 1 and the MM page address field is set to the contents of NODE(K,3). For both paging-in and paging-out, the contents of register TMR are stored into the translation memory TM. Thus, the translation memory word is updated for virtual address translation. Finally, control is returned to the user program.

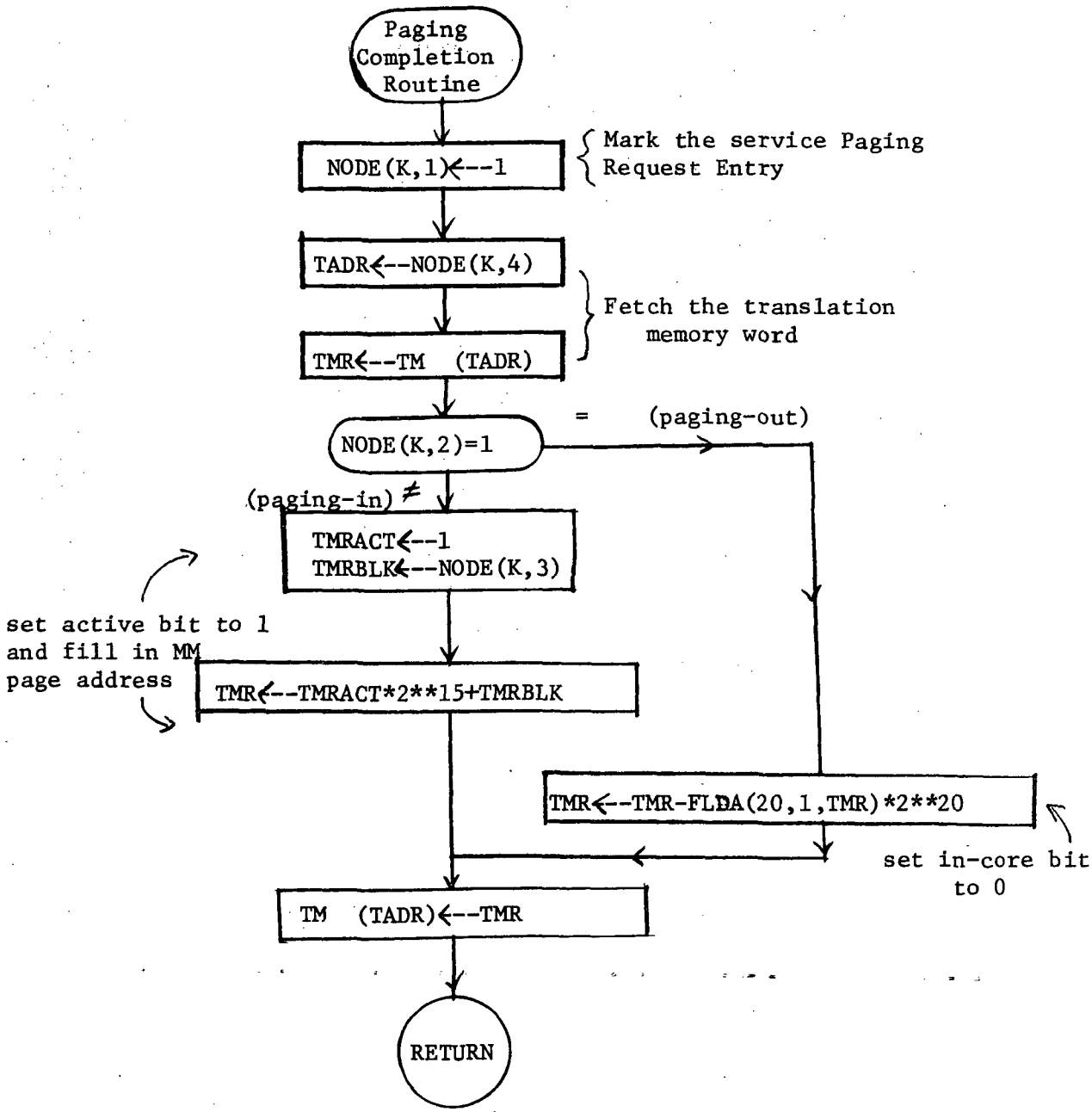


Fig. 21 Flow chart of the Paging Completion (IOCS2) Routine which updates the Translation Memory

6. Simulation of the Page Transfer Operation Under IOCSP

The paging drum channel was simulated by using Simula on the UNIVAC 1108 and reported in reference (5). This simulation, also in Simula, is an extended model which combines the PDC and the IOCSP. We first describe the simulation program, then the inputs, and finally the results and discussions.

Two assumptions have been made. First, the drum has 8 sectors instead of 16 sectors. Second, the translation memory has 512 words instead of 1,024 words.

6.1 Simulation program

The structure of the program simulating the IOCSP and the PDC is shown in the flow chart of Fig. 22. When the simulation begins, data are loaded into the counters for the paging request entries, for the paging request queue and for the IOCSP simulation routines. Data are also loaded into the translation memory, the paging drum memory, and the main memory. Next, a paging request is read from a card reader. The IOCS1 routine is then called to handle the incoming paging request. It examines the request, determines its function, and constructs a paging request entry. The UNIT routine is next called to service a page-transfer request. It constructs a channel program (for the PDC) in main memory locations 4, 5, 6, and 7. The FETCH routine, is then called to transfer the channel program in the main memory to the PDC.

The paging drum channel PDC is initiated to execute the channel program. Upon completion of a page-transfer, the IOCS2 routine is called. It marks the paging request entry as "serviced" and updates the translation memory for address translation. If there are more requests, the next paging request is read and processed; otherwise, the simulation terminates.

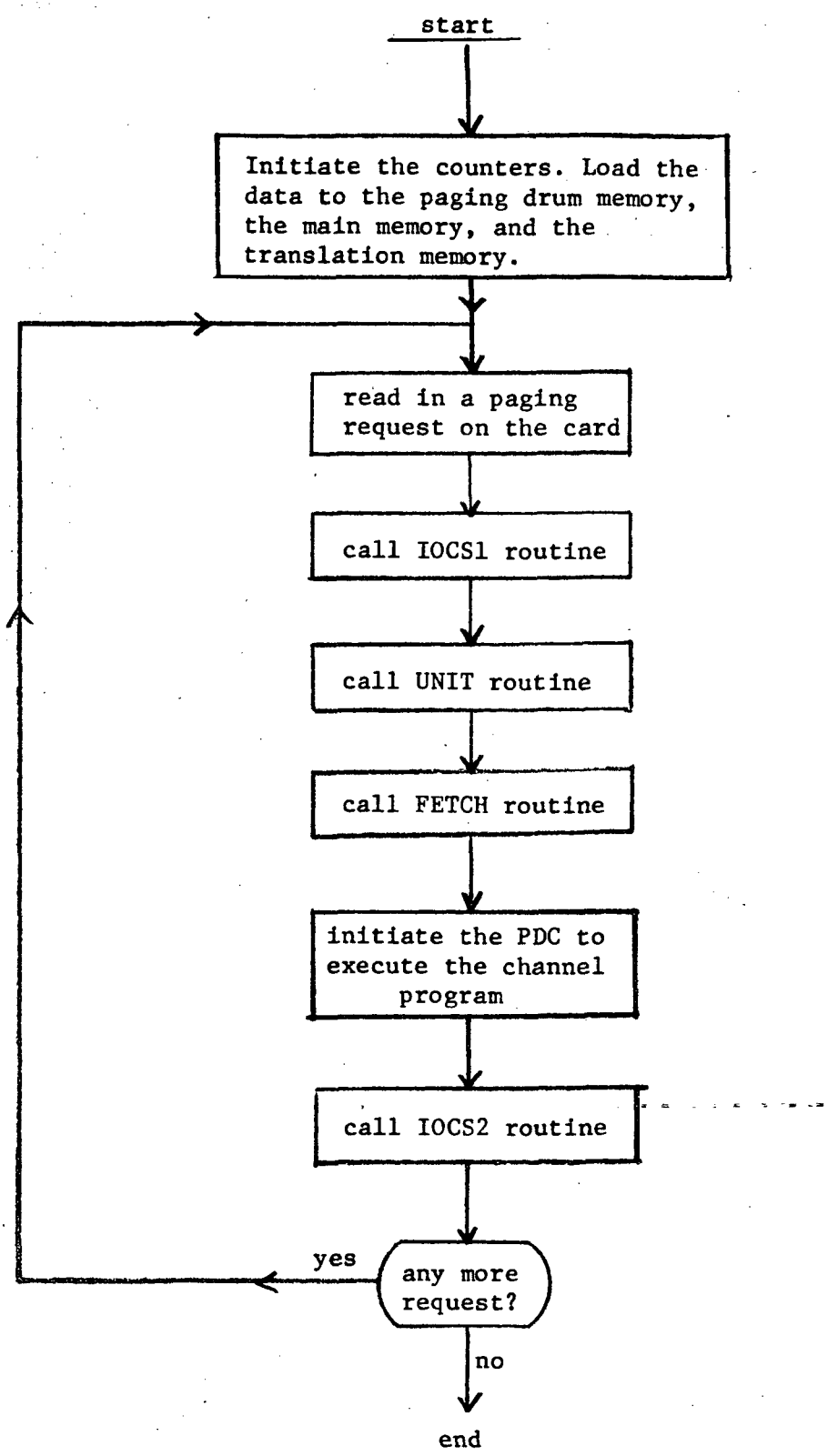


Fig. 22, Flow chart showing the structure of the simulation program

6.2 Inputs

Two sets of input data are chosen. The first set consists of 8 paging-out requests, while the second set consists of 8 paging-in requests. A paging request is specified by the following:

(a) paging instruction (1 for paging-out and 0 for paging-in);

(b) main memory page address (between 0 and 63);

(c) virtual page address (between 0 and 511).

Table 3 contains 8 paging-out requests of Test Data 1. As shown, the paging instructions are all set to 1. Main memory page 32 is to be transferred to virtual page 32; main memory page 39 is to be transferred to virtual page 39. Table 4 shows the data initially in the main memory pages. With Test Data 1, 8 main memory pages are paged out to the 5th field of the paging drum as shown in Fig. 23. In this figure, main memory page 32 is stored at the first drum page of sector 0, and main memory page 39 is stored at the first drum page of sector 7.

After the paging-out operations, the contents of drum sector 0 through 7 are tabulated in Table 5. As shown, there are eight 32's in the drum sector 0; there are eight 33's in drum sector 1, etc. These eight drum pages are then paged-in by Test Data 2.

Table 6 contains the 8 paging-in requests of Test Data 2. As shown, the paging instructions are all set to 0. Virtual page 32 is to be transferred to main memory page 32; and virtual page 39 is to be transferred to main memory page 39. With Test Data 2, 8 drum pages are paged in as illustrated in Fig. 24. In the figure, the first drum page of sector 0 is transferred to the 32nd main memory page; and the first drum page sector 7 is transferred to the 39th main memory page.

Table 3: Paging-out requests of Test Data 1

(for paging-out 8 MM pages to 8 drum sectors)

Test	Paging* Instruction	Main Memory Page Address	Virtual Page Address
1	1	32	032
2	1	33	033
3	1	34	034
4	1	35	035
5	1	36	036
6	1	37	037
7	1	38	038
8	1	39	039

*Paging-out when 1

Table 4 Initial Data of 8 Main Memory Pages of Test Data 1

MM Page number	MM word number	Contents of words	MM page number	MM word number	Contents of words
32	0 to 7	32	36	0 to 7	36
33	0 to 7	33	37	0 to 7	37
34	0 to 7	34	38	0 to 7	38
35	0 to 7	35	39	0 to 7	39

Table 5 Data on the 8 Drum Sectors of Test Data 2

(for paging-in 8 MM pages from the 8 Drum Sectors)

Drum Sector Number	Field Number	Word Address	Contents of Word	Drum Sector Number	Field Number	Word Address	Contents of Word
0	5	0 to 7	32	4	5	0 to 7	36
1	5	0 to 7	33	5	5	0 to 7	37
2	5	0 to 7	34	6	5	0 to 7	37
3	5	0 to 7	35	7	5	0 to 7	38

Table 6 Paging-in requests of Test Data 2

(for paging-in 8 MM pages from 8 drum sectors)

Test	Paging* Instruction	Main Memory Page Address	Virtual Page Address
1	0	32	032
2	0	33	033
3	0	34	034
4	0	35	035
5	0	36	036
6	0	37	037
7	0	38	038
8	0	39	039

*Paging-in when 0

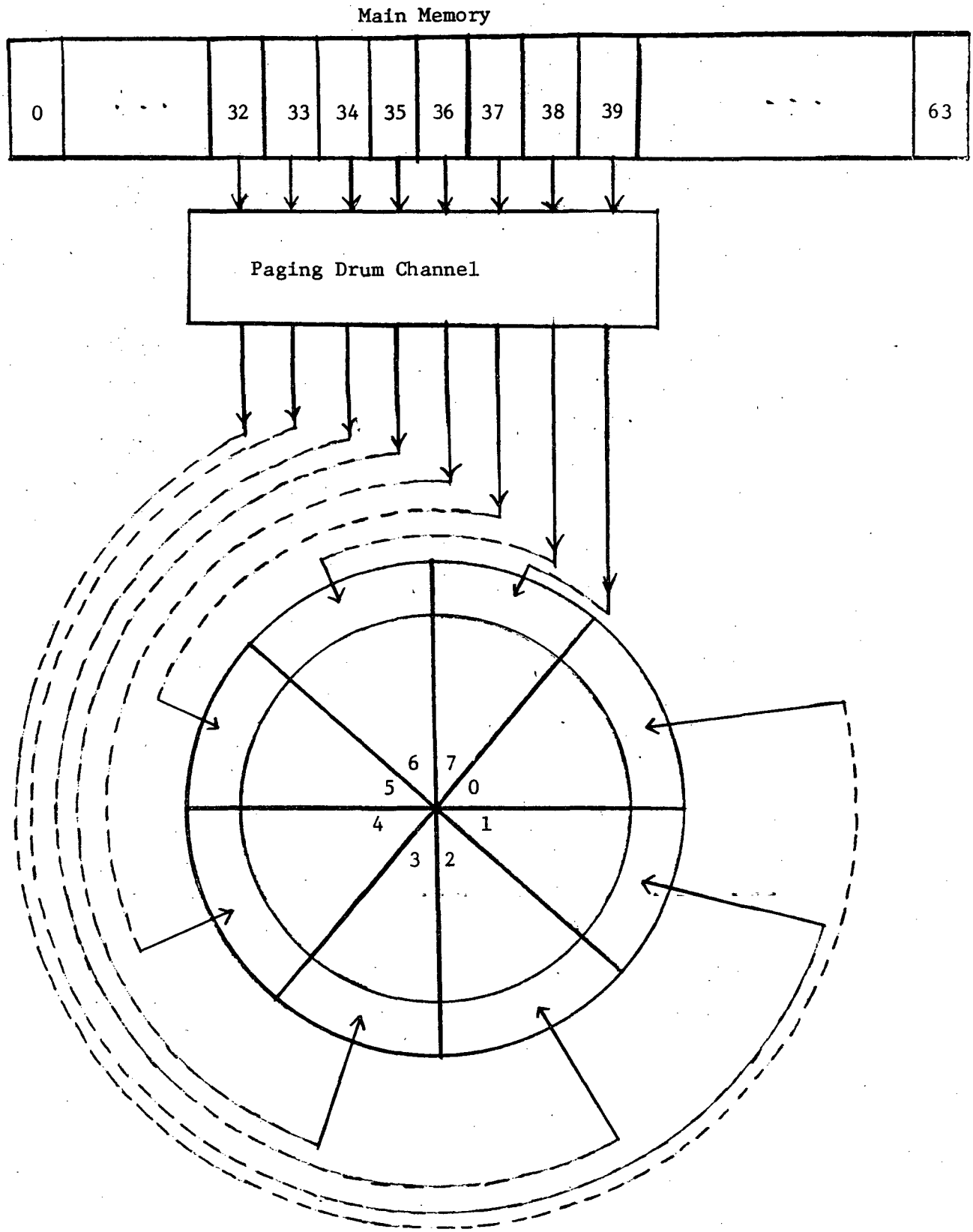


Fig. 23 Paging-out 8 pages from Main Memory to 8 drum sectors (Test Data 1)

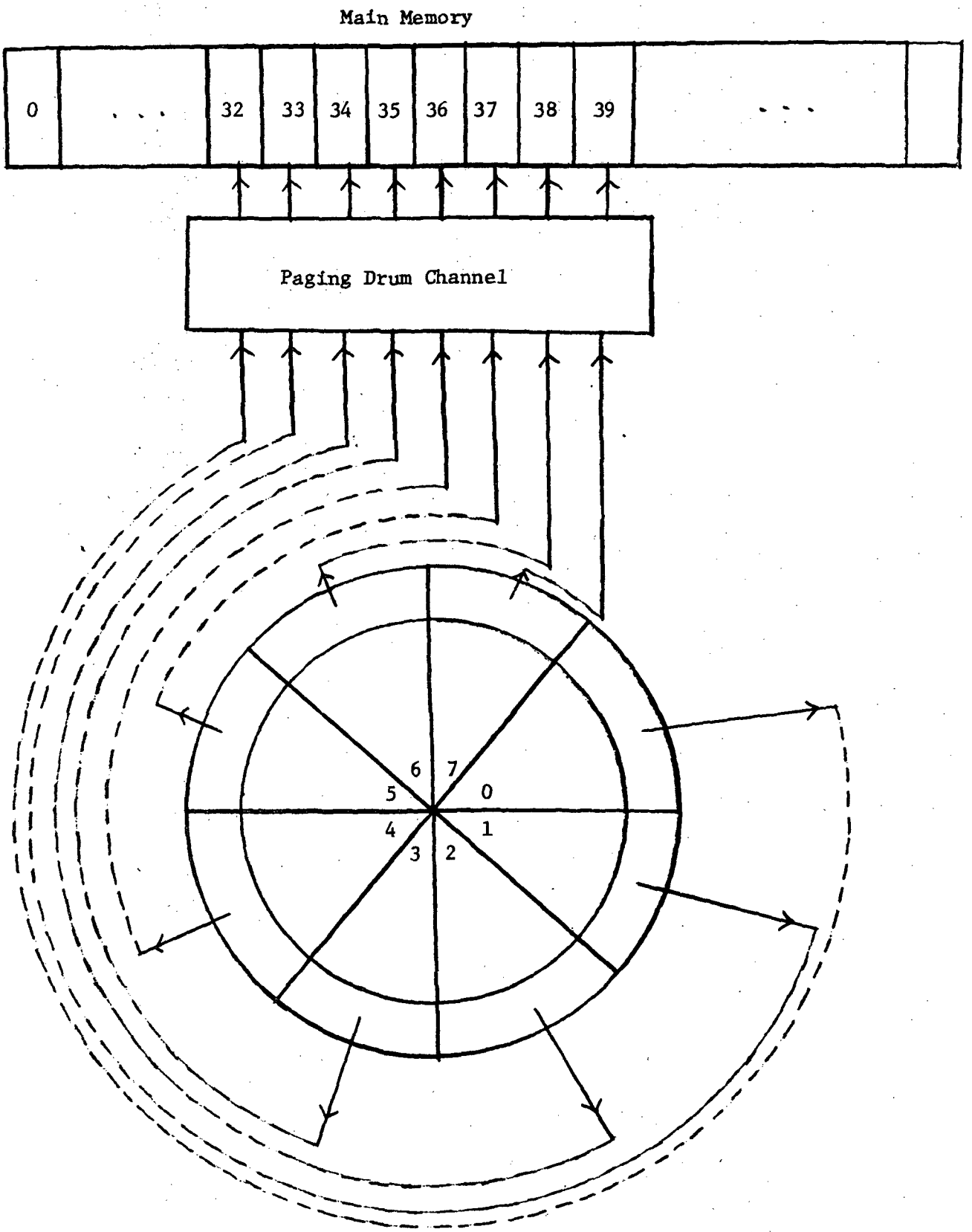


Fig. 24 Paging-in 8 pages into the Main Memory from 8 drum sectors (Test Data 2)

6.3 Results

There are two kinds of simulation output,

(a) the contents of the main memory buffer register, the drum buffer register, the word count in the page, and the page transfer status after a word is transferred.

(b) the modified channel command word, the listheads, the drum field address, and the drum sector address of the page after a page is transferred.

Table 7 explains the variables in the printed outputs. Register PAGEPOST contains the main memory page address of the posted page; the page to be transferred. Register RW is the paging-in (when 1) or paging-out (when 0) indicator. Register SECTOR contains the drum sector address (there are 8 drum sectors). Register FIELD contains the drum field address (there are 64 drum fields). Register SBR2 is the main memory buffer register, while register DBR is the drum memory buffer register. Register COUNT is a counter which specifies the word address of the current page being transferred. COM(0,1) and COM(0,2) store the left half and the right half of the channel command word for the 0th drum sector respectively. LISTS(0) contains the listheads of the 0th sector queue in the page table memory. INTERRUPT(PAGE) is the page-transfer-complete indicator; it is set to 1 when the page-transfer is successful. PTRAN is the page-transfer indicator. When it is 1 or 2, it represents a paging-in and paging-out operations respectively; when it is 3, it indicates an error. COM(7,1) and COM(7,2) store the left half and the right half of the CCW for the 7th drum sector respectively. LISTS(7) contains the listheads of the 7th sector queue in the page table memory.

The outputs for paging-out 8 main memory pages to 8 drum sectors of Test Data 1 are summarized in Table 8-(a) through 8-(d). Table 8-(a)

Table 7 Variables in the Print-out

Variables	Representation
PAGEPOST	Register which contains the MM page address of the posted page
RW	Paging-in or paging-out indicator
SECTOR	Drum sector address
FIELD	Drum field address
SBR2	Main memory buffer register
DBR	Drum memory buffer register
COUNT	Counter of the word address in the current page
COM(0,1)	Left half of the channel command word for the 0th drum sector
COM(0,2)	Right half of the channel command word for the 0th drum sector
LISTS(0)	List-heads of the 0th sector queue
INTERRUPT(PAGE)	Page-transfer-complete indicator
PTRAN	Page-transfer status indicator
COM(7,1)	Left-half of the channel command word for the 7th drum sector
COM(7,2)	Right-half of the channel command word for the 7th drum sector
LISTS(7)	List-heads of the 7th sector queue

Table 8 Octal Output from Test Data 1

(a) Print out when the first page is being transferred

PAGEPOST	RW	SECTOR	FIELD	SBR2	DBR	COUNT
40	1	0	5	40	40	0
40	1	0	5	40	40	1
40	1	0	5	40	40	2
40	1	0	5	40	40	3
40	1	0	5	40	40	4
40	1	0	5	40	40	5
40	1	0	5	40	40	6
40	1	0	5	40	40	7

(b) Print out after the first page is transferred

COM(0,1)	COM(0,2)	LISTS(0)	INTERRUPT(PAGE)	PTRAN
140540	40	4040	1	2

(c) Print out when the last page is being transferred

PAGEPOST	RW	SECTOR	FIELD	SBR2	DBR	COUNT
47	1	7	5	47	47	0
47	1	7	5	47	47	1
47	1	7	5	47	47	2
47	1	7	5	47	47	3
47	1	7	5	47	47	4
47	1	7	5	47	47	5
47	1	7	5	47	47	6
47	1	7	5	47	47	7

(d) Print-out after the last page is transferred

COM(7,1)	COM(7,2)	LISTS(7)	INTERRUPT(PAGE)	PTRAN
140547	47	4747	1	2

shows the output when main memory page 40_8 is transferred. In Table 8-(a), PAGEPOST contains 40_8. When a word of page 40_8 is transferred, SBR2 is 40_8 and DBR is 40_8. COUNT is the word counter for the first page; it varies from 0 to 7. RW is 1 indicating a paging-out operation. Since SECTOR is 0 and FIELD is 5, the first page 40_8 is transferred to field 5 in drum sector 0.

After page 40_8 is paged-out, the print out of the corresponding CCW, the listheads for sector queue 0, the page transfer interrupt signal, and the page transfer status are summarized in Table 8-(b). In this table, COM(0,1) is $\underline{1405}40_8$ where the underlined portion $\underline{1405}_8$ or 11000001_2 which is interpeted as follows: (a) the C field of the CCW is 1 implying a page-transfer, (b) the RW field is 1 implying a paging-out operation, (c) the drum field address is 5; and the remaining 40_8 not underlined is the main memory page address of the current page. COM(0,2) contains 40_8 which is the first actual word of the first page. The listheads of sector queue 0 are 40_8 and 40_8; thus LISTS(0) is 4040_8. INTERRUPT(PAGE) is 1 indicating that page transfer is successfully completed. PTRAN is 2 indicating a paging-out operation.

Table 8-(c) shows the output after a word of main memory page 47_8. In this Table, PAGEPOST is 47_8. Each time when there is a word transfer, SBR2 and DBR become 47_8 and 47_8 respectively. COUNT contains 0 through 7. RW is 1, indicating a paging-out operation. SECTOR is 7 and FIELD is 5 so that the last page is transferred to field 5 in drum sector 7.

After paging-out page 47_8, the print out of the corresponding CCW, the listheads for sector queue 7, the page transfer interrupt signal, and the page transfer status are summarized in Table 8-(d). COM(7,1) is $\underline{1405}47_8$ where the underlined portion $\underline{1405}_8$ has the same meaning as that for the first page, and the remaining 47_8 not underlined is the main memory page address of the current page. The listheads of sector queue 7 are 47_8 and 47_8;

therefore, LISTS(7) is 4747_8. The settings for INTERRUPT(PAGE) and PTRAN are the same as those for the first page.

The outputs for paging-in the 8 main memory pages of Test Data 2 mentioned previously are summarized in Table 9-(a) to 9-(d). Table 9-(a) and 9-(b) differ only bh the value for RW. In Table 9-(a), RW is 0, indicating a paging-in operation. Table 9-(b) and 8-(b) differ only by the values for COM(0,1) and PTRAN. In Table 9-(b), COM(0,1) is $\underline{100540}_8$ where the $\underline{1005}_8$ is 100000001_2 which is interpreted as follows: (a) the C field of the CCW being 1 indicates a page transfer; (b) it is paging-in; and (c) the drum field address is 5; and the remaining 40_8 not underlined is the main memory page address of the current page. In Table 9-(b), PTRAN is set to 1 for paging-in.

Table 9-(c) and Table 8-(c) differ only by the value for RW. In Table 9-(c), RW is 0, indicating a paging-in operation.

Table 9-(d) and Table 8-(d) differ by the values for COM(7,1) and PTRAN. In Table 9-(d), COM(7,1) is $\underline{100547}_8$ where the underlined portion $\underline{1005}_8$ has the same meaning as that for the first page. In Table 9-(d), PTRAN is set to 1 for paging-in.

6.4 Discussion

Fig. 25 shows the relationships of the IOCSP to the processing program and the paging drum channel system (model 1). In this figure, the dashed box contains a pagin drum channel which controls a large-capacity paging drum. The IOCSP is the interface between the processing program and the paging drum channel system. The IOCSP handles the page-transfer requests issued by the processing program. A paging-in request causes a page of words to be transferred from the paging drum through the paging drum channel to the main memory. A paging-out request will cause a page of words to be

Table 9 Octal Output from Test Data 2

(for paging-in 8 drum pages from 8 drum sectors)

(a) Print-out when the first page is being transferred

PAGEPOST	RW	SECTOR	FIELD	SBR2	DBR	COUNT
40	0	0	5	40	40	0
40	0	0	5	40	40	1
40	0	0	5	40	40	2
40	0	0	5	40	40	3
40	0	0	5	40	40	4
40	0	0	5	40	40	5
40	0	0	5	40	40	6
40	0	0	5	40	40	7

(b) Print-out after the first page is transferred

COM(0,1)	COM(0,2)	LISTS(0)	INTERRUPT(PAGE)	PTRAN
100540	40	4040	1	1

(c) Print-out when the last page is being transferred

PAGEPOST	RW	SECTOR	FIELD	SBR2	DBR	COUNT
47	0	7	5	47	47	0
47	0	7	5	47	47	1
47	0	7	5	47	47	2
47	0	7	5	47	47	3
47	0	7	5	47	47	4
47	0	7	5	47	47	5
47	0	7	5	47	47	6
47	0	7	5	47	47	7

(d) Print-out after the last page is transferred

COM(7,1)	COM(7,2)	LISTS(7)	INTERRUPT(PAGE)	PTRAN
100547	74	4747	1	1

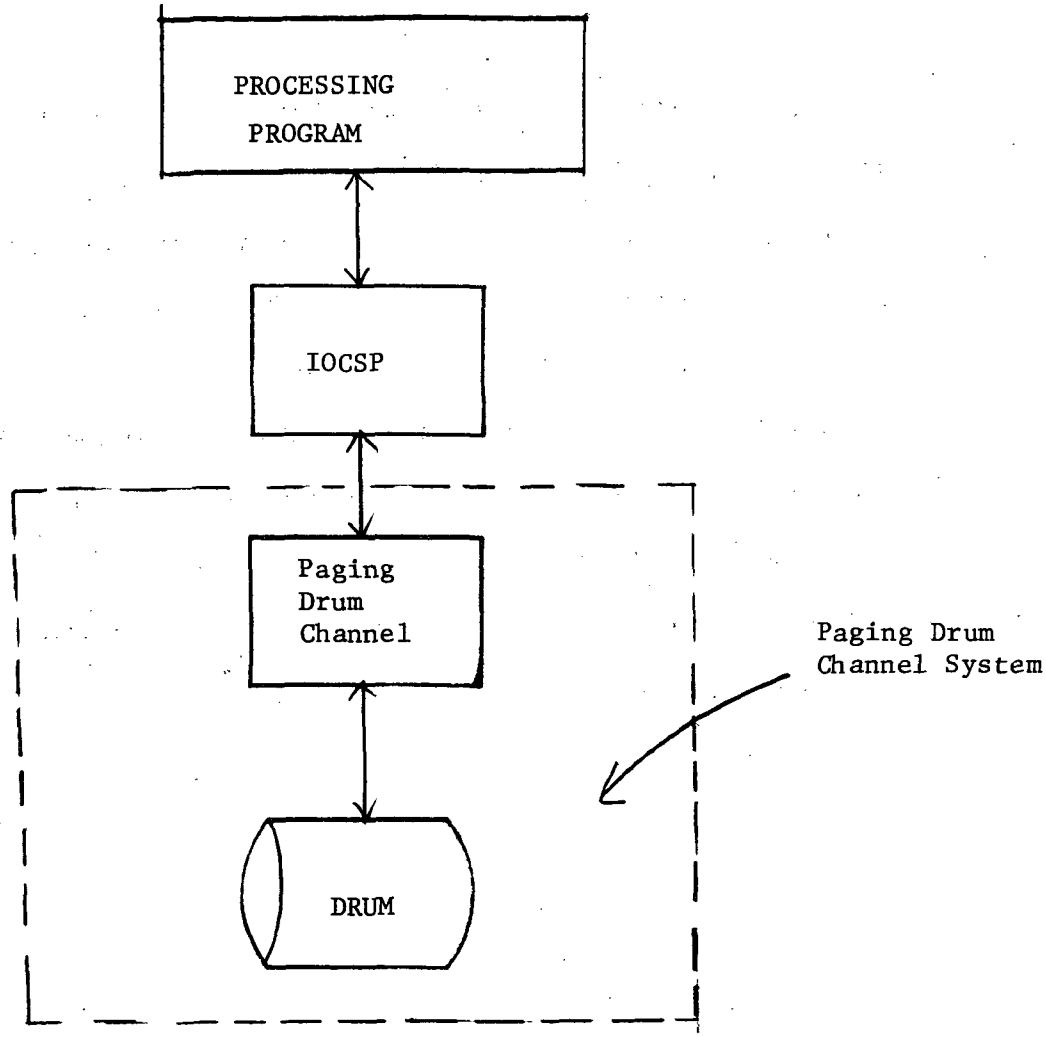


Fig. 25 The relationship of IOCSP to the Processing Program and the paging drum channel system (model 1)

transferred from the main memory through the paging drum channel to the paging drum.

Fig. 26 shows the relationship of the Input-Output Control System IOCS which includes the IOCSP and the three channels with five I/O devices attached to the channels (model 2). As shown, multiplexor channel 1 and multiplexor channel 2 are controlled by the IOCS while the Paging Drum Channel System is controlled by the IOCSP. With these two additional multiplexor channels, the IOCS in Fig. 26 may consist of many routines. For example, the I/O Scheduling Routine selects the channel and the I/O device; consequently, one unit interpretive routine is required for each type of I/O devices. The I/O Buffering Routine simulates double buffering for reading ahead from and writing ahead to magnetic tapes.

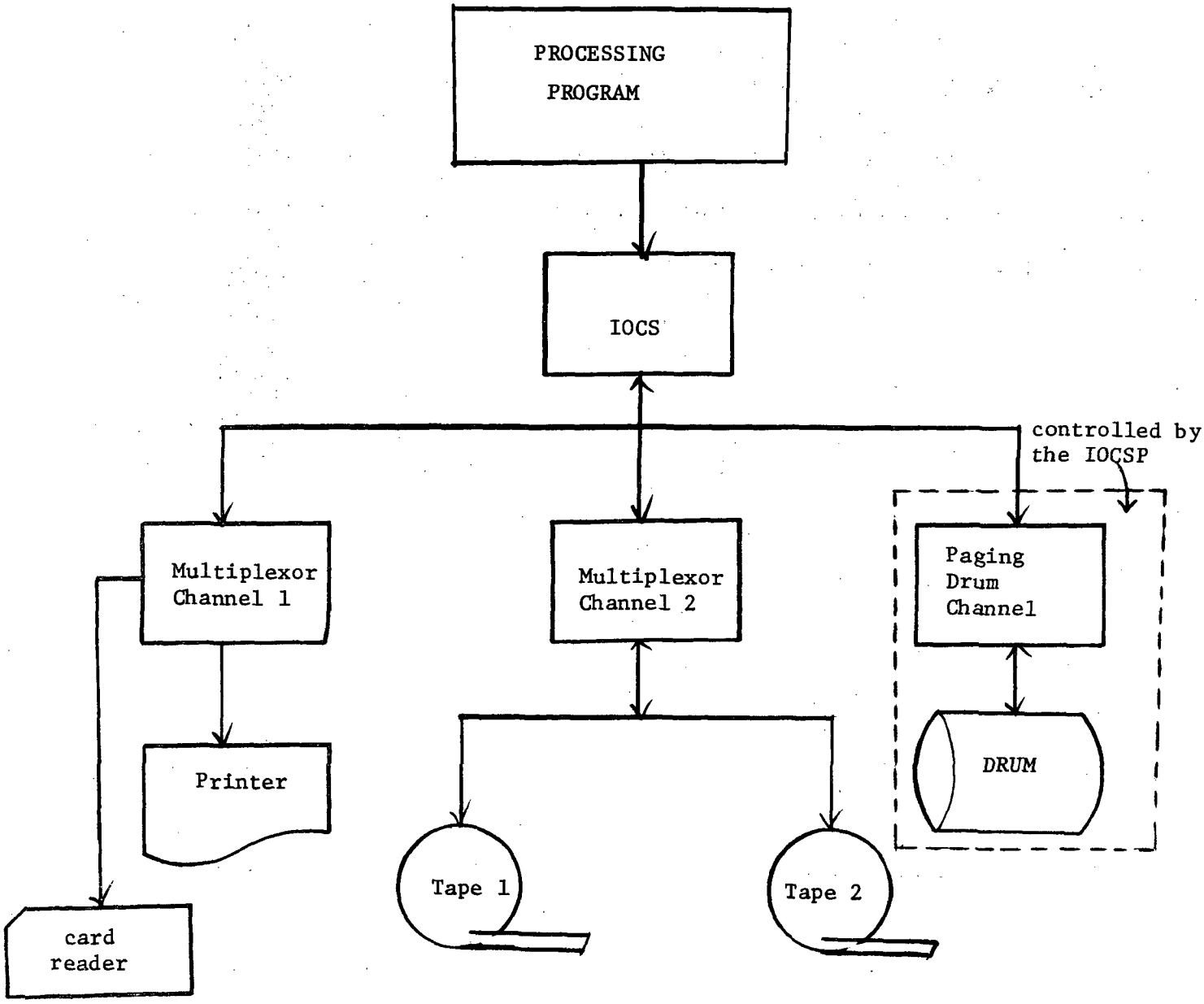


Fig.26 The relationship of IOCS to the Processing Program and three channels with five I/O devices (model 2)

7. Acknowledgment

The authors wish to express their thanks to O. R. Pardo and Jeffrey Yeh for thier helpful discussions and suggestions and to Nancy A. Nowell for her typing of the entire manuscript.

8. References

1. Chu, Y., "Introduction to Computer Organization", Prentice-Hall, Inc., 1970.

2. Chu, Y., "Notes on Channel Organization", Computer Science Center, University of Maryland, November, 1970.

3. Coffman, E.G., Jr., "Analysis of a Drum Input/Output Queue Under scheduled Operation in a Paged Computer System", JACM, January 1969, pp. 73-90.

4. IBM S. R. L., IBM 7094/7094 Input/Output Control System, File Number 7090-30, Form C28-6345.

5. Kwok, G., "Simulation of a Paging Drum Channel", Technical Report TR-155, University of Maryland, May, 1971.

6. Pardo, O. R., "A Virtual Memory System Design", Technical Report 71-144, Computer Science Center, University of Maryland, January 1971.

7. RCA Information Systems, Spectra 70 System 70/61, Processor Reference Manual, 70-61-601, September 1969.

8. Yeh, J.W., "An IOCS Algorithm for Microprogramming", Technical Report, 70-124, University of Maryland, July 1970.

9. Denning, "The Working Set Model for Program Behavior", CACM, May 1968, pp. 323-333.

10. Denning, Peter J., "Virtual Memory", Computer Surveys, Vol. 2, No.3, September 1970.

11. Kilburn, T., et. al., "One-level Storage System", IRE Trans. on EC., April, 1962, pp. 223-235.

12. Weingarten, A., "The Eschenback Drum Scheme", CACM, July, 1966, pp. 509-512.

13. Denning, P. J., Effects of scheduling on file memory operation", Proc. of SJCC, 1967, pp. 9-21.

14. Denning, P. J., "Queuing models for file memory operations", Report MAC-TR-21 (thesis), Project MAC, M.I.T., Cambridge, Mass., October, 1965.

APPENDIX A. SIMULATION PROGRAM LISTING

```
RUN A3,001-11-768,KWOK,1,100
ALG,CIS        PDCSIM,PDCSIM
 EXTERNAL NON-RECURSIVE INTEGER PROCEDURE FLDA $
 SIMULA BEGIN
     COMMENT  ***** INPUT-OUTPUT CONTROL SYSTEM *** $
 COMMENT INDEX FOR THE FIELDS IN A PAGING REQUEST ENTRY $
     INTEGER J $
 COMMENT INSTRUCTION ** 1 FOR PAGING OUT , 0 FOR PAGING IN $
     INTEGER INST $
 COMMENT MAIN MEMORY PAGE ADDRESS $
     INTEGER MMPAGE $
 COMMENT TRANSLATION MEMORY ACCRESS FOR THE VIRTUAL PAGE $
     INTEGER TMADDR $
 COMMENT 40 PAGING REQUEST IS MAXIMUM
 NODE(K,1) = MARK BIT FOR THE REQUEST
 NODE(K,2) = PAGING INSTRUCTION, 0 FOR PAGING-IN AND 1 FOR PAGING-OUT
 NODE(K,3) = MM PAGE ADDRESS
 NODE(K,4) = VIRTUAL PAGE ADDRESS
 NODE(K,5) = DRUM FIELD ADDRESS
 NODE(K,6) = DRUM SECTOR ADDRESS
 ONLY NODE(K,2), NODE(K,3), NODE(K,4) ARE INPUT FROM THE USER $
     INTEGER ARRAY NODE(1..40,1.. 10) $
     COMMENT EACH NODE REPRESENTS A PAGING REQUEST ENTRY $
     COMMENT TRANSLATION MEMORY SUBSYSTEM $
     INTEGER ARRAY TMEM(0..511 )$
     COMMENT TRANSLATION MEMORY ADDRESS REGISTER $
     INTEGER TADR $
     COMMENT TRANSLATION MEMORY BUFFER REGISTER $
     INTEGER TMR $
     COMMENT VIRTUAL ADDRESS REGISTER $
     INTEGER VAD $
     COMMENT MAIN MEMORY ADDRESS REGISTER (FOR CPU ) $
     INTEGER MADR1 $
     COMMENT READ/WRITE REGISTER ( FOR CPU) $
     INTEGER RW1 $
     COMMENT READ ERROR FLAG $
     INTEGER RFLAG $
     COMMENT WRITE ERROR FLAG $
     INTEGER WFLAG $
     COMMENT PAGE FAULT REGISTER $
     INTEGER PFAULT $
     COMMENT MAIN MEMORY ACCESS REGISTER $
     INTEGER MA1$
     COMMENT STORAGE BUFFER REGISTER ( FOR THE CPU )$
     INTEGER SBR1 $
     COMMENT PAGE IS ACTIVE WHEN 1 $
     INTEGER TMRACT $
     COMMENT PAGE HAS BEEN REFERENCED WHEN 1 $
     INTEGER TMRREFD$
     COMMENT PAGE HAS BEEN CHANGED WHEN 1 $
     INTEGER TMRCHGD $
     COMMENT PAGE IS WRITE PROTECTED WHEN 1 $
     INTEGER TMRWP $
     COMMENT PROTECTION KEY OF THE PAGE $
```

```
      INTEGER IMRWKEY $
A2    COMMENT MM PAGE ADDRESS FOR THE VIRTUAL PAGE IF IT IS IN THE MM $
      INTEGER IMRBLK $
      COMMENT VIRTUAL PAGE ADDRESS $
      INTEGER VADPAGE $
      COMMENT VIRTUAL WORD ADDRESS $
      INTEGER VADWORD $
COMMENT ************************************************************$
  COMMENT ***A PAGING DRUM CHANNEL ***$
COMMENT FOR TESTING AND SIMULATION ON THE 1108 MEMORY,
        PAGE SIZE IS REDUCED TO 8 WORDS PER PAGE
        64 PAGES IN THE MAIN MEMORY
         512 PAGES IN THE PAGING DRUM WHICH HAS  8 SECTORS
        64 DRUM PAGES IN A DRUM SECTOR
        36 BITS PER WORD$
  COMMENT K IS THE INDEX OF THE PAGING REQUEST ENTRY $
      INTEGER K $
COMMENT I IS THE LOOP CONTROL VARIABLE$
  INTEGER I$
COMMENT MAIN MEMORY AND RELATED REGISTERS$
COMMENT MAIN MEMORY$
  INTEGER ARRAY MEM(0..63,0..7)$
COMMENT MAIN MEMORY ADDRESS REGISTER$
  INTEGER MAR$
COMMENT MAIN MEMORY BUFFER REGISTER$
  INTEGER MBR$
COMMENT MAIN MEMORY READ/WRITE CONTROL REGISTER$
  INTEGER RW2$
COMMENT MAIN MEMORY PAGE ADDRESS$
    INTEGER MADR2BLOCK$
COMMENT MAIN MEMORY BUFFER REGISTER$
  INTEGER SBR2$
COMMENT MA(2) IS MAIN MEMORY ACCESS REGISTER$
  INTEGER MA2$
COMMENT MADR2(WRD) IS MAIN MEMORY WORD ADDRESS$
  INTEGER MADR2WRD$
COMMENT MAIN MEMORY PAGE ADDRESS OF THE POSTED PAGE$
 INTEGER  PAGEPOST$
COMMENT PAGE-TABLE MEMORY$
  INTEGER ARRAY PAGETABLE(0..63,1..2)$
COMMENT PAGE-TABLE MEMORY ADDRESS REGISTER$
  INTEGER PADR$
  COMMENT PAGE TABLE BUFFER REGISTER $
      INTEGER ARRAY PTR (1 .. 2) $
COMMENT PAGE-TABLE MEMORY BUFFER REGISTER$
  INTEGER ARRAY PTR2(1..2)$
  COMMENT SUBFIELDS OF A PAGE DESCRIPTOR $
  INTEGER PTR2CH$
  INTEGER PTR2SEC$
  INTEGER PTR2ROW$
  INTEGER PTR2LB$
 INTEGER PTR2LF$
COMMENT LISTHEAD MEMORY$
  INTEGER ARRAY LISTS(0.. 7)$
COMMENT LISTHEAD MEMORY ADDRESS REGISTER$
  INTEGER SECTORS$
COMMENT LISTHEAD MEMORY BUFFER REGISTER$
  INTEGER PTL$
COMMENT POINTER THE FIRST PAGE OF A SECTOR QUEUE IN THE DRUM$
  INTEGER PTLFP$
```

```
COMMENT POINTER THE  LAST PAGE OF A SECTOR QUEUE IN THE DRUM$
  INTEGER PTLLP$
COMMENT GPTL IS AN AUXILIARY REGISTER$
  INTEGER GPTL$
  INTEGER GPTLFP$
  INTEGER GPTLLP$
COMMENT PAGE TABLE SEMAPHOR$
  INTEGER PTSEM1$
  INTEGER PTSEM2$
COMMENT PAGING DRUM AND RELATED REGISTER$
COMMENT PAGING DRUM MEMORY$
  INTEGER ARRAY PDRUM(0.. 7,1.. 64,0..7)$
COMMENT PAGING DRUM SECTOR ADDRESS$
  INTEGER CWORDSECT$
COMMENT PAGING DRUM FIELD ADDRESS $
  INTEGER FIELD $
COMMENT PAGING DRUM FIELD WORD ADDRESS $
  INTEGER CWORDCOUNT$
COMMENT DRUM READ/WRITE CONTROL REGISTER$
    INTEGER RW$
COMMENT DRUM BUFFER REGISTER$
    INTEGER DBR$
COMMENT DRUM ACTIVE INDICATOR$
    INTEGER DACTV$
COMMENT COMMAND MEMORY AND RELATED REGISTERS$
    INTEGER ARRAY COM(0.. 7,1..2)$
COMMENT COMMAND MEMORY ADDRESS REGISTER$
    INTEGER SEC$
COMMENT COMMAND MEMORY BUFFER REGISTER$
    INTEGER ARRAY COMMAND(1..2)$
COMMENT SUBREGISTERS OF THE COMMAND WORD$
    INTEGER COMC$
    INTEGER COMRWC$
    INTEGER COMCHAN$
    INTEGER COMPGE$
    INTEGER COMFIRSTWORD$
COMMENT DRUM BUFFER STATUS REGISTER$
  INTEGER BS$
COMMENT INTERRUPT(DRUMPAGE)$
  INTEGER INTERRUPTDP$
COMMENT MAIN MEMORY PAGE WHICH INTERRUPT OCCURED$
   INTEGER PAGINT$
COMMENT INTERRUPT(PAGE)$
  INTEGER INTERRUPTPGE$
COMMENT CURRENT PAGE ADDRESS$
  INTEGER PC$
COMMENT WORD COUNT OF THE PAGE$
  INTEGER COUNT$
COMMENT PAGE TRANSFER DIRECTION,0 WHEN NO TRANSFER,1 WHEN DRUM TO MEMORY,
            2 WHEN MEMORY TO DRUM,3 WHEN ERROR OCCURS$
  INTEGER PTRAN$
COMMENT PAGE TRANSFER COMPLETE WHEN 1$
  INTEGER PAGEI$
COMMENT PAGE POSTING INDICATOR$
  INTEGER POST$
  FORMAT F1 ( '  **  TRANSFER A PAGE  **********************',A3.3)$
  FORMAT F3 ( X2,I10.8,2X, I10.8, 2X, I10.8, A1.1)$
  FORMAT F4 (' COMMAND IN OCTAL =', I10.8,A1,1)$
      FORMAT F5(A, I1,I2,I3)$
```

```
A4    FORMAT F20( X8 ,' LISTHEADS IN OCTAL', A1.3)$
      FORMAT F21 (X5,I2,X2,I10.8, A1.1)$
      FORMAT F22(X2 ,' CHANNEL COMMAND WORDS IN OCTAL',A1.3)$
        LOCAL LABEL LAST$
      LOCAL LABEL L3$
    COMMENT ************************************************************$
          ACTIVITY IOCS1$
          BEGIN
          FORMAT F2(E, X25,' *** SIMULATION OF IOCS *** ',A2.2) $
    COMMENT PRINT TITLE $
          WRITE(F2) $
      IF INST EQL 1 THEN WRITE('PAGING-OUT') ELSE WRITE('PAGING-IN')$
          K = K +1 $
      IF K GTR 40 THEN K = 1 $
      COMMENT CONSTRUCTION OF A PAGING REQUEST ENTRY $
    COMMENT MARK BIT IS ZERO FIRST AND IS SET TO 1 WHEN REQUEST IS SERVICED$
          NODE(K,1) = 0 $
    COMMENT 0 FOR PAGING IN AND 1 FOR PAGING OUT $
          NODE(K,2) = INST $
    COMMENT MAIN MEMORY PAGE ADDRESS $
          NODE(K,3) = MMPAGE $
    COMMENT TRANSLATION MEMORY ADDRESS $
          NODE(K,4) = TMADDR $
    COMMENT CONSTRUCTION OF A CHANNEL COMMAND WORD IN MM LOCATIONS 4,5 $
      NODE(K,5) = 1 $
      COMMENT COMPUTE DRUM PAGE ADDRESS $
      IF TMADDR LSS 8 THEN GO TO N1 $
    NO .. TMADDR = TMADDR - 8 $
      IF TMADDR LSS 0 THEN GO TO N2 $
      COMMENT COMPUTE DRUM FIELD ADDRESS $
      NODE(K,5) = NODE(K,5) + 1 $
        GO TO NO $
    N1 .. NODE(K,5) = 1 $
         NODE(K,6) = TMADDR $
         GO TO N3 $
      COMMENT COMPUTE DRUM SECTOR ADDRESS $
    N2 .. NODE(K,6) = TMADDR + 8 $
    N3 .. HOLD(50) $
          TERMINATE (CURRENT) $
          END $
    COMMENT ************************************************************$
    COMMENT UNIT INTERPRETIVE ROUTINE FOR THE PAGING DRUM CHANNEL $
    ACTIVITY UNIT $
      BEGIN
                MBR = 8**5 + NODE(K,2)*4*8**4 + NODE(K,5)*8**2 + NODE(K,3)$
          MEM(0,4) = MBR $
    COMMENT COMPUTE COMMAND (2) $
                MBR = NODE(K,3) $
          MEM(0,5) = MBR $
          COMMENT CONSTRUCTION OF A PAGE DESCRIPTOR WORD IN MM LOCATION 6,7$
      COMMENT COMPUTE PAGE DESCRIPTOR FOR THE MM PAGE $
                MBR = 0 $
          MEM(0,6) = MBR $
      COMMENT COMPUTE PTR (2) $
                MBR = NODE(K,5)*8**3*4 + NODE(K,6)*(8**2)*2 + 8**2 $
          MEM(0,7) = MBR $
          COMMENT OUTPUT CHANNEL PROGRAM $
      HOLD(50)$
      TERMINATE (CURRENT)$
      END$
```

```
COMMENT ************************************************************$
  ACTIVITY FETCH $
    BEGIN
COMMENT FETCH THE CHANNEL PROGRAM FROM THE MAIN MEMORY BUILT BY IOCS $
    MAR = 4 $
    MBR = MEM(0,4) $
    SBR2 = MBR $
    COMMAND(1) = SBR2 $
    MAR = 5 $
    MBR = MEM(0,5) $
    SBR2 = MBR $
    COMMAND(2) = SBR2 $
    SEC = NODE( K,6) $
    COM(SEC,1) = COMMAND(1) $
    COM(SEC,2) = COMMAND(2) $
    MAR = 6 $
    MBR = MEM(0,6) $
    SBR2 = MBR $
    PTR2(1) = SBR2 $
    PTR(1) = PTR2(1) $
    MAR = 7 $
    MBR = MEM(0,7) $
    SBR2 = MBR $
    PTR2(2) = SBR2 $
    PTR(2) = PTR2(2) $
    PADR = NODE(K,3) $
    PAGETABLE(PADR,1) = PTR(1) $
    PAGETABLE(PADR,2) = PTR(2) $
  CWORDSECT=-1 $
    HOLD(50) $
  TERMINATE( CURRENT) $
  END $
COMMENT ************************************************************$
  ACTIVITY IOCS2 $
  BEGIN
      LOCAL LABEL POUT $
COMMENT MARK THE PAGING REQUEST ENTRY AS SERVICED $
    NODE(K,1) = 1 $
COMMENT FIX UP THE TRANSLATION MEMORY $
    TADR = NODE(K,4) $
    TMR = TMEM(TADR) $
COMMENT FETCH A WORD FROM THE TRANSLATION MEMORY INTO THE BUFFER $
COMMENT CHECK PAGING OUT / PAGING IN $
    IF NODE(K,2) EQL 1 THEN GO TO POUT $
COMMENT SET PAGE IN CORE BIT TO 1 $
    TMRACT = 1 $
COMMENT FILL IN MM PAGE ADDRESS INTO THE TRANSLATION MEMORY ENTRY $
    TMRBLK = NODE(K,3) $
    TMR = TMRACT*2**15 + TMRBLK $
    GO TO OUT $
COMMENT PAGING OUT OPERATION COMES HERE $
COMMENT TURN IN CORE BIT OFF $
POUT .. TMRACT = 0 $
    TMR = TMR - FLDA(20,1,TMR)*2**20 $
COMMENT STORE THE MODIFIED TRANSLATION MEMORY WORD $
OUT .. TMEM(TADR) = TMR $
    HOLD(50) $
  TERMINATE( CURRENT) $
  END $
```

```
COMMENT ************************************************************$
  ACTIVITY LEFT$
  BEGIN
  WRITE( ' *** DRUM READ/WRITE SUBSEQUENCE *** ')$
COMMENT  CHECK PAGE SWAP INDICATOR$
  IF DACTV EQL 1 THEN GOTO L1$
  PAGEI=1$
  PTRAN=0$
  WRITE('PTRAN=',PTRAN)$
  HOLD(5.0)$
  TERMINATE( CURRENT)$
COMMENT PAGE TRANSFER STARTS HERE $
COMMENT  BRANCH OUT TO WRITE.$
L1 .. IF RW EQL 1 THEN GO TO L5 $
  MA2=0$
COMMENT
  WRITE('BS= ',BS,' RW= ',RW)$
COMMENT  DRUM TO MAIN MEMORY TRANSFER( READ BRANCH)$
L4.. IF MA2 EQL 1 THEN GO TO L4$
  BS=1$
L6.. IF BS EQL 0 THEN GOTO L6$
  MADR2WRD= COUNT$
  WRITE( '  INPUT WORD COUNT = ',MADR2WRD)$
  DBR = PDRUM( SEC, FIELD, CWORDCOUNT) $
  WRITE( ' DBR =', DBR)$
COMMENT INPUT FROM THE PAGING DRUM ONE WORD$
  SBR2=DBR$
  RW2=RW$
  MA2=1$
    BS = 0 $
  WRITE('*************************DRUM TO MEMORY')$
  MBR= SBR2$
  WRITE( ' MBR ', MBR)$
  MEM(MADR2BLOCK,MADR2WRD)= MBR$
  IF COUNT EQL 7     THEN BEGIN
    PAGEI = 1$
      DACTV=0$
  END$
L7 ..     IF PAGEI EQL 0 THEN BEGIN
      COUNT= COUNT+1$
  CWORDCOUNT= CWORDCOUNT +1$
      GOTO L1$
  END$
  COUNT=COUNT+RW$
  IF COUNT EQL 7     THEN BEGIN
  PTRAN=RW+1$
  WRITE('PTRAN = ',PTRAN)$
  HOLD(320.0)$
  TERMINATE( CURRENT)$
 GOTO  L3$
  END
ELSE BEGIN
COMMENT  SET ERROR INDICATOR$
  INTERRUPTDP=1$
  PTRAN=3$
  WRITE('PTRAN =', PTRAN)$
  HOLD(320.0)$
  TERMINATE( CURRENT)$
  GOTO  L3$
  END$
```

```
COMMENT  MAIN MEMORY TO DRUM TRANSFER ( WRITE BRANCH)$
L5.. MA2=0$
  IF MA2 EQL 1 THEN GOTO L5$
  RW2= RW$
  MADR2WRD= COUNT+1$
  WRITE( ' WORD COUNT = ', MADR2WRD)$
  MA2=1$
L8.. MA2=0$
  IF MA2 EQL 1 THEN GOTO L8$
L9.. BS =1$
COMMENT DATA TRANSFERS FROM MEMORY BUFFER TO DRUM BUFFER$
  SBR2= MEM(MADR2BLOCK,MADR2WRD)$
  WRITE('SBR2 =',SBR2)$
  WRITE( '*********************************************MEMORY TO DRUM')$
  DBR=SBR2$
      BS = 0 $
  WRITE('DBR= ',DBR)$
  PDRUM( SEC, FIELD,CWORDCOUNT) = DBR $
COMMENT IN THE WRITE OPERATION THE WORD COUNT DOES NOT INCLUDE THE
        FIRST WORD OF THE PAGE. IF EXACTLY 7 MORE WORDS (0-6)
        WERE WRITTEN, AN ENTIRE PAGE WILL BE COMPLETELY TRANSFERRED$
  IF COUNT EQL 6    THEN BEGIN
  PAGEI = 1$
  DACTV= 0$
  END$
  GOTO L7 $
COMMENT THE SECOND PARALLEL PROCESS STARTS HERE$
COMMENT  CHECK IF THE READ/WRITE LOOP NEEDED.$
  END$
COMMENT ************************************************************$
ACTIVITY RIGHT $
  BEGIN
COMMENT MEMORIES UPDATING SUBSEQUENCE STARTS HERE$
L10 .. SECTORS = SEC$
      WRITE( ' *** UPDATING SUESEQUENCE *** ')$
  PTL= LISTS(SECTORS)$
  PTLFP=FLDA(24,6,PTL)$
  PTLLP= FLDA(30,6,PTL)$
COMMENT  FOR EMPTY QUEUE, SET PAGE SWAPPING INDICATOR$
  IF PTLFP EQL 0 THEN BEGIN
  CCMC=0$
  GOTO L13$
  END$
COMMENT GETTING A PAGE$
K1.. PTSEM1=0$
  IF PTSEM1 EQL 1 THEN GOTO K1$
  PTSEM2=1$
COMMENT  PUT THE LIST HEAD INTO REGISTER GPTL$
  GPTL=PTL$
  GPTLFP=FLDA(24,6,GPTL)$
  GPTLLP= FLDA(30,6,GPTL)$
  WRITE('GPTL(FP=)',GPTLFP,' GPTL(LP)=', GPTLLP)$
  PADR=GPTLFP$
  PC= GPTLFP$
COMMENT  GET A PAGE DESCRIPTOR FROM THE PAGE TABLE MEMORY$
  PTR2(1)= PAGETABLE(PADR,1)$
  PTR2(2)= PAGETABLE(PADR,2)$
  PTR2CH = FLDA(17,8,PTR2(2))$
  WRITE( ' FIELD ADDRESS = ', PTR2CH)$
```

```
A8    PTR2SEC= FLDA(25,4,PTR2(2))$
      WRITE( ' SECTOR = ', PTR2SEC)$
      PTR2ROW= FLDA(29,1,PTR2(2))$
      PTR2LB= FLDA(2,6,PTR2(1))$
      PTR2LF= FLDA(8,6,PTR2(1))$
      GPTLFP= PTR2LF$
    COMMENT  TRANSFER THE UPDATED LIST HEAD TO PTL$
      PTL = GPTL$
      PTSEM2= 0$
    COMMENT  STORE THE UPDATED LISTHEAD TO THE LISTS MEMORY$
    L15.. LISTS(SECTORS)= PTL$
    COMMENT  SET UP A CHANNEL COMMAND WORD$
         COMC=1$
         COMPGE=PC$
         COMCHAN=PTR2CH$
         COMRWC= PTR2ROW$
      COMMAND(1)= COMC*8**5+COMRWC*4*8**4+COMCHAN*8**2+ COMPGE$
      IF PTR2ROW EQL 0 THEN GOTO L13$
    COMMENT  WRITE ONTO DRUM$
    L11.. MA2= 0$
     IF MA2 EQL 1 THEN GOTO L11$
      RW2=1$
    COMMENT  SET UP MAIN MEMORY ADDRESS REGISTER$
      MADR2BLOCK= PC$
      MADR2WRD= 0$
      MA2=1$
    L12 .. MA2=0$
      IF MA2 EQL 1 THEN GOTO L12$
    COMMENT SET UP THE FIRST DATA WORD IN THE COMMAND REGISTER$
      MBR= MEM(MADR2BLOCK,0)$
      SBR2= MBR$
      COMMAND(2)= SBR2$
    COMMENT  PUT THE NEW CHANNEL COMMAND WORD INTO THE COMMAND MEMORY$
    L13.. COM(SEC,1)= COMMAND(1)$
          COM(SEC,2)= COMMAND(2)$
      WRITE(F22) $
      WRITE( SEC, COM(SEC,1), COM(SEC,2),F3)$
      PAGEI = 0$
      POST = 1$
    L14.. IF PAGEI EQL 1 THEN BEGIN
      HOLD(50)$
      TERMINATE (CURRENT)$
      END$
      POST= 1$
    COMMENT REQUEST - ACCEPT SUBSEQUENCE STARTS HERE  *****$
      IF POST EQL 0 THEN GOTO L14$
    COMMENT  LOAD PAGE DESCRIPTOR SEQUENCE STARTS HERE$
    K2 .. PTSEM1 = 0$
      IF PTSEM1 EQL 1 THEN GOTO K2$
      PTSEM2=1$
    COMMENT  GET A PAGE DESCRIPTOR$
      PADR= PAGEPOST$
      PTR2(1)= PAGETABLE(PADR,1)$
      PTR2(2)= PAGETABLE(PADR,2)$
    COMMENT  FREE THE PAGE-TABLE MEMORY$
      PTSEM2=0$
      PTR2SEC= FLDA(25,4,PTR2(2))$
      SECTORS= PTR2SEC$
      PTL= LISTS(SECTORS)$
      PTLFP= FLDA(24,6,PTL)$
```

```
COMMENT  BRANCH TO PUT THE PAGE DESCRIPTOR IN THE PTM $
  IF PTLFP EQL 0 THEN GOTO K3$
COMMENT  GET THE NEXT CHANNEL COMMAND WORD$
  SEC= SECTORS$
  COMMAND(1)= COM(SEC,1)$
  COMMAND(2)= COM(SEC,2)$
  IF COMC EQL 0 THEN BEGIN
  PC= PAGEPOST$
    POST = 0 $
  PAGEI = 1$
  GOTO L15$
  END$
COMMENT  PUT A PAGE BACK TO THE SECTOR QUEUE IN THE PTM $
K3.. PTSEM1=0$
  IF PTSEM1 EQL 1 THEN GOTO K3$
  PTSEM2=1$
  GPTL= PTL$
  GPTLFP= FLDA(24,6,GPTL)$
COMMENT  IF THE SECTOR QUEUE IS EMPTY THE CURRENT PAGE BECOMES THE FIRST
         PAGE (FRONT OF THE QUEUE)$
  IF GPTLFP EQL 0 THEN BEGIN
  GPTLFP = PAGEPOST$
  GPTL = GPTLFP*8**2 + FLDA(30,6,GPTL)$
  GOTO K4$
  END$
COMMENT INSERT THE NEW PAGE AT THE REAR OF AN NON-EMPTY QUEUE$
  GPTLLP= FLDA(30,6,GPTL)$
  PADR= GPTLLP$
  PTR2(1)= PAGETABLE(PADR,1)$
  PTR2(2)= PAGETABLE(PADR,2)$
COMMENT UPDATE A PAGE DESCRIPTOR$
  PTR2LF= MOD( PTR2(1),2**28)$
  PTR2LB = PTR2(1) - PTR2LF* 2**28 $
  PTR2(1)= PTR2LB + PAGEPOST*2**22$
  PAGETABLE(PADR,1)= PTR2(1)$
  PAGETABLE(PADR,2)= PTR2(2)$
COMMENT  GET THE PAGE DESCRIPTOR OF THE POSTED PAGE$
K4.. PADR = PAGEPOST$
  PTR2(1)= PAGETABLE(PADR,1)$
  PTR2(2)= PAGETABLE(PADR,2)$
COMMENT  LET THE BACKWORD POINTER POINT TO THE REAR OF THE QUEUE$
  PTR2LB= GPTLLP$
  PTR2LF =0$
  PTR2(1)= GPTLLP*2**28$
COMMENT  RETURN THE NEW PAGE DESCRIPTOR TO THE PAGE-TABLE MEMORY$
  PAGETABLE(PADR,1)= PTR2(1)$
  PAGETABLE(PADR,2)= PTR2(2)$
COMMENT UPDATE LISTHEADS$
  GPTLLP= MOD( GPTL,8**2)$
  GPTLFP= GPTL - GPTLLP$
  GPTL = GPTLFP + PADR$
  PTL= GPTL$
  PTSEM2=0$
COMMENT  STORE LISTHEAD$
  LISTS(SECTORS)= PTL$
  WRITE( ' SECTOR    LISTHEADS')$
  WRITE( SECTORS, LISTS(SECTORS),F21)$
  POST=0$
  PAGEI = 1$
```

```
A10   GOTO L14$
      END$
   COMMENT ************************************************************$
     COMMENT MAIN ACTIVITY STARTS HERE $
   COMMENT CLEAR TRANSLATION MEMORY $
       FOR J = 0 STEP 1 UNTIL 511  DO
       TMEM(J) = 0 $
         COMMENT COUNTER $
         K= 0 $
   COMMENT INITIALIZE COUNTER $
     I= 1 $
   COMMENT INITIALIZATION OF THE MAIN MEMORY $
     FOR MAR = 1 STEP 1 UNTIL 63 DO BEGIN
       FOR COUNT = 0 STEP 1 UNTIL 7 DO
       MEM(MAR, COUNT) = MAR $
     END $
     COMMENT INITIALIZATION OF THE PAGING DRUM MEMORY $
     COMMENT EACH DRUM PAGE CONTAINS THE VALUE OF ITS VIRTUAL PAGE NUMBER$
     FOR FIELD = 1 STEP 1 UNTIL 64 DO BEGIN
       FOR SEC = 0 STEP 1 UNTIL  7 DO BEGIN
       FOR COUNT = 0 STEP 1 UNTIL 7 DO
       PDRUM (SEC,FIELD,COUNT) = TMADDR $
       TMADDR = TMADDR + 1 $
       END $
       END $
     WRITE(' 1 FOR PAGING-OUT , 0 FOR PAGING IN ')$
     WRITE(' 2 DIGITS FOR MM PAGE ,3 DIGITS FOR VIRTUAL PAGE ADDRESS ')$
     WRITE(' 032032 MEANS TO PAGE IN DRUM PAGE 32 TO MM PAGE 32')$
     WRITE(' 132032 MEANS TO PAGE OUT MM PAGE 32 TO DRUM PAGE 32 ')$
     COMMENT INPUT A PAGING REQUEST $
   XIOCS .. READ( INST,MMPAGE,TMADDR,F5,LAST )$
            WRITE(' *** INPUT PAGING REQUEST *** ')$
         ACTIVATE NEW IOCS1$
   COMMENT CALL IN UNIT INTERPRETIVE ROUTINE $
         ACTIVATE NEW UNIT$
     COMMENT FETCH CHANNEL PROGRAM FROM THE MAIN MEMORY $
     ACTIVATE NEW FETCH $
     HOLD( 1000)$
   PDC.. PAGEI=1$
     IF PAGEI EQL 0 THEN BEGIN
     HOLD(1.0)$
     GOTO PDC$
     END ELSE
   L3 .. PAGINT=MADR2BLOCK$
         INTERRUPTPGE=1$
     HOLD (10.0)$
   L0..     PAGEI=0$
     CWORDSECT= CWORDSECT+1$
   COMMENT THERE ARE 8 PAGE-TRANSFERS IN ONE DRUM REVOLUTION $
    IF CWORDSECT EQL  8 THEN CWORDSECT = 0 $
   COMMENT  OBTAIN A CHANNEL COMMAND WORD$
   COMMENT INPUT FROM CARD IS A COMMAND WORD$
     COUNT=0$
     CWORDCOUNT=0$
     WRITE(F1)$
     COMMAND(1)=COM(SEC,1)$
     WRITE( COMMAND(1), F4)$
     COMMAND(2)=COM(SEC,2)$
     WRITE( ' FIRSTWORD = ', COMMAND(2))$
   COMMENT DECODING A COMMAND WORD AND PUT THE CONTROL INFORMATION
```

```
        INTO THE APPROPRIATE REGISTERS$
  CCMC=FLDA(20,1,COMMAND(1))$
COMMENT  INDICATE A PAGE HAS BEEN SWAPPED WHEN 1$
  DACTV= CCMC$
  CCMRWC=FLDA(21,1,COMMAND(1))$
  RW= CCMRWC$
  CCMCHAN=FLDA(22,8,COMMAND(1))$
  FIELD = COMCHAN $
  CCMPGE=FLDA(30,6,COMMAND(1))$
  MADR2BLOCK=CCMPGE$
  PAGEPOST = CCMPGE$
  WRITE(' PAGEPOST ',PAGEPOST)$
COMMENT DATA TRANSFER$
  CCMFIRSTWORD = COMMAND(2)$
COMMENT  TRANSFER THE FIRST WORD OF A PAGE TO THE DRUM BUFFER REGISTER$
     DBR= COMFIRSTWORD$
  WRITE( 'FIELD ADDRESS = ', FIELD)$
  IF RW EQL 1 THEN BEGIN
  WRITE( 'WRITE OPERATION,RW= ',RW)$
  COMMENT OUTPUT FIRST WORD TO DRUM$
  WRITE( '*********************************************MEMORY TO DRUM!')$
COMMENT FOR WRITE OPERATION THE FIRST WORD IS ALREADY IN THE BUFFER
        BEFORE ENTERING THE WRITE LOOP $
  WRITE( ' WORD COUNT = ', COUNT)$
  WRITE(' DBR= ',DBR)$
  PDRUM (SEC, FIELD,0) = DBR $
  END ELSE
                    WRITE( ' READ OPERATION, RW= ',RW)$
COMMENT  PARALLEL PROCESS STARTS HERE$
  HOLD (40.0)$
  ACTIVATE NEW LEFT$
  ACTIVATE NEW RIGHT$
  HOLD( 1000)$
  ACTIVATE NEW IOCS2 $
  HOLD(1000) $
  GO TO XIOCS $
LAST .. END $
 ASM,IS          DECODE,DECODE
$(1)     AXR$.
.         ROUTINE FOR DECODING PAGE DESCRIPTORS,CHANNEL COMMAND WORDS
.         AND LISTHEADS FOR THE PAGING OPERATION .
.         INPUT FORMAT  ***  FLDA(I,J,K)
FLDA*   LA        A2,*3,X11      . FETCH THE WORD K                    $
        LA        A0,*1,X11      . FETCH THE BIT POSITION I
        SA,H2     A0,L1          . GETTING RID OF UPPER BITS
        SA,H2     A0,L2          .
L1      LSSL      A2,0           . SHIFT LEFT I BIT POSITIONS
L2      SSL       A2,0           . SHIFT RIGHT I BIT POSITIONS
        LA,U      A1,36          . COMPUTE 36-I-J
        AN        A1,*2,X11      .
        AN        A1,A0          .
        SA,H2     A1,L3          . GETTING RID OF THE LOWER BITS
L3      SSL       A2,0           . SHIFT RIGHT 36-I-J BIT POSITIONS
        J         4,X11          . RETURN TO THE ALGOL SIMULATION PROGRAM
        END
 MAP
 XQT
132032                                  TEST DATA 1 ... PAGING-OUT 8 PAGES
133033
```

```
A12  134C34
     135C35
     136C36
     137C37
     138C38
     139C39
     032C32                                  TEST DATA 2 ...PAGING-IN 8 PAGES
     033C33
     034C34
     035C35
     036C36
     037C37
     038C38
     039C39
      FIN
```

TOTAL CARDS = 671